Quasi Deterministic Radio Channel Generator User Manual and Documentation

Document Revision: v1.2.3-307

April 30, 2014

Fraunhofer Heinrich Hertz Institute
Wireless Communication and Networks
Einsteinufer 37, 10587 Berlin, Germany

e-mail: quadriga@hhi.fraunhofer.de
http://www.quadriga-channel-model.de

Fraunhofer
Heinrich Hertz Institute

Contributors

Editor:	Fraunhofer Heinrich Hertz Institute Wireless Communication and Networks Einsteinufer 37, 10587 Berlin, Germany
Contributing Authors:	Stephan Jaeckel, Leszek Raschkowski, Kai Börner and Lars Thiele *Fraunhofer Heinrich Hertz Institute* Frank Burkhardt and Ernst Eberlein *Fraunhofer Institute for Integrated Circuits IIS*

Grants and Funding

This work was supported by

- the European Space Agency (ESA) in the Advanced Research in Telecommunications Systems (ARTES) programme under contract AO/1-5985/09/08/NL/LvH (Acronym: MIMOSA), [EHB+13]
  http://telecom.esa.int/telecom/www/object/index.cfm?fobjectid=31061
- the German Federal Ministry of Economics and Technology (BMWi) in the national collaborative project IntelliSpektrum under contract 01ME11024
  http://www.intellispektrum.de

Acknowledgements

The authors thank G. Sommerkorn, C. Schneider, M. Kaeske [Ilmenau University of Technology (IUT), Ilmenau, Germany] and V. Jungnickel [Heinrich Hertz Institute (HHI), Berlin, Germany] for the fruitful discussions on the QuaDRiGa channel model and the manuscript of this document.

How to Cite this Document

[JRB+14] S. Jaeckel, L. Raschkowski, K. Börner, L. Thiele, F. Burkhardt and E. Eberlein, "QuaDRiGa - Quasi Deterministic Radio Channel Generator, User Manual and Documentation", Fraunhofer Heinrich Hertz Institute, Tech. Rep. v1.2.3-307, 2014.

Contents

eMail: quadriga@hhi.fraunhofer.de

List of Figures

List of Tables

List of Acronyms

AoA	angle of arrival. 54, 68
AoD	angle of departure. 54
BS	base station. 11, 14, 35, 38, 54, 55, 71
CIR	channel impulse response. 23, 33, 42, 60, 62, 94
DS	delay spread. 14, 59–62, 72, 73
FIR	finite impulse response. 60
KF	Ricean K-factor. 59, 62, 63, 71
LBS	last bounce scatterer. 66, 67
LHCP	left hand circular polarized. 25, 27, 58
LOS	line of sight. 11–13, 23, 30, 37, 38, 42, 43, 48, 51, 54, 55, 63, 66–68, 70–72, 80, 81, 83, 86–88, 91, 94–97, 99, 100, 103–105, 109–111
LSP	large scale parameter. 11, 13, 14, 16, 17, 23, 32, 36, 38, 39, 41, 44, 48, 59–61, 66, 72, 74, 90, 94, 97, 99, 104, 109
MIMO	multiple-input multiple-output. 10, 11, 13, 15, 23, 38, 44, 56
MIMOSA	MIMO over Satellite. 12
MPC	multipath component. 12, 54, 71
MSE	mean square error. 28
MT	mobile terminal. 11, 13, 42, 44, 54, 55, 71, 72
NLOS	non line of sight. 11, 13, 23, 42, 49, 50, 62, 66, 69, 70, 80, 81, 83, 85, 87, 91, 92, 94, 97, 99, 101, 105, 107, 109–111
PDP	power delay profile. 62
PG	path gain. 32, 36, 71
PL	path loss. 49
RHCP	right hand circular polarized. 27, 58
Rx	receiver. 63, 66–68, 71
SCM	spatial channel model. 11, 56, 66, 69
SF	shadow fading. 71
SISO	single input single output. 10
SSG	state sequence generator. 14
Tx	transmitter. 63, 66–68, 71
UML	unified modeling language. 19
WINNER	Wireless World Initiative for New Radio. 10, 11, 23, 42, 48, 54, 59, 66, 70, 72
WSS	wide sense stationary. 13
WSSUS	wide sense stationary uncorrelated scattering. 12
XPR	cross polarization ratio. 23, 42, 51, 70

List of Symbols

γ	Polarization rotation angle derived from the NLOS XPD
$\hat{\phi}_m$	Offset angle of the m-th subpath relative to (ϕ, θ)
κ	Cross polarization power ratio
λ	Carrier wavelength
$\mathbf{a}$	Departure- or Arrival vector in Cartesian Coordinates
$\mathbf{B}$	Autocorrelation map of a large scale parameter
$\mathbf{e}_r$	Position of the r^{th} element in the receive array (relative to the array center)
$\mathbf{F}(\phi, \theta)$	Antenna field pattern
$\mathbf{G}_l$	Coefficient matrix of a tap between all n_t Tx antennas and n_r Rx antennas
$\mathbf{H}_{n,s}$	MIMO channel matrix for all n_t Tx antennas and n_r Rx antennas in frequency domain
$\mathbf{M}$	Polarization coupling matrix
$\mathbf{o}_{r/t}$	Antenna orientation vector for the receiver/transmitter
$\mathbf{p}_{r/t}$	Projection of the orientation vector $\mathbf{o}_{r/t}$ on the plane perpendicular to $\mathbf{r}$
$\mathbf{r}$	Wave travel direction in Cartesian coordinates
ϕ	Azimuth angle, ϕ^a for arrival (AoA), ϕ^d for departure (AoD)
ϕ_{LOS}	LOS direction seen from the receiver
$\psi_{l,m,s}$	Phase of the m^{th} subpath of the l^{th} cluster at snapshot position s
ρ	Correlation coefficient
σ_ϕ	Angular Spread
σ_τ	RMS delay spread
$\tau_{l,s}$	Delay of the l^{th} cluster at snapshot position s
PL	Path Loss (linear scale), $\text{PL}_{[\text{dB}]} = 10 \cdot \log_{10} \text{PL}$ is for logarithmic scale
SD	Sample Density in units of samples per half wave length
SF	Shadow Fading (linear scale), $\text{SF}_{[\text{dB}]} = 10 \cdot \log_{10} \text{SF}$ is for logarithmic scale
θ	Elevation angle, θ^a for arrival (EoA), θ^d for departure (EoD)
ϑ	Polarization rotation angle used for the coupling matrix $\mathbf{M}$
ζ	Per Cluster Shadow Fading
a_k, b_k	Filter coefficients
c	Speed of Light
$C_\phi(L, K)$	Correction function for the initial angles
$C_\tau(K)$	Correction function for the initial path delays
c_{AoA}	Cluster-wise azimuth spread of arrival
d	Distance; also used a path length
d_λ	Decorrelation distance
d_r	Total length of the wave travel direction vector $\mathbf{r}$
d_{LOS}	Distance between the receiver position and the scatterers for the LOS cluster
d_{px}	Distance (in m) between two adjacent pixels of the autocorrelation map $\mathbf{B}$
f_c	Carrier Frequency
f_S	Sampling Rate – in units of samples per meter
f_T	Sampling Rate – in units of samples per second
$g_{r,t,l}$	Complex amplitude of a tap between Tx antenna t and Rx antenna r
$h_{r,t,n}$	Channel coefficient in frequency domain
K	Ricean K-factor (linear scale), $K_{[\text{dB}]}$ is for logarithmic scale
k	Filter coefficient index
L	Number of clusters or paths

eMail: quadriga@hhi.fraunhofer.de

l	Cluster or path index, $l = 1...L$
m	Subpath index, $m = 1...20$
N	Number of carriers
n	Carrier index; $n = 1 \dots N$
n_r	Number of receive antennas
n_t	Number of transmit antennas
P_l	Cluster power
r	Receive antenna index; $t = 1 \dots n_r$
r_τ	Delay distribution proportionality factor
S	Number of snapshots in one segment
s	Snapshot index within one segment
t	Transmit antenna index; $t = 1 \dots n_t$
v	Speed
X,Y,Z	Random variables, e.g. $X \sim \text{uni}(X_{\min}, X_{\max})$ for uniform or $X \sim \text{N}(\mu, \sigma)$ for Normal distribution

eMail: quadriga@hhi.fraunhofer.de

References

[3GP05] 3GPP TDOC R4-050854. Spatial radio channel models for systems beyond 3G. Technical report, Elektrobit, Nokia, Siemens, Philips, Alcatel, Telefonica, Lucent, Ericsson, 8 2005.

[3GP11] 3GPP TR 25.996 v10.0.0. Spatial channel model for multiple input multiple output (MIMO) simulations. Technical report, 3 2011.

[BHS05] D.S. Baum, J. Hansen, and J. Salo. An interim channel model for beyond-3G systems. *Proc. IEEE VCT '05 Spring*, 5:3132–3136, 2005.

[CG03] Xiaodong Cai and Georgios B. Giannakis. A two-dimensional channel simulation model for shadowing processes. *IEEE Trans. Veh. Technol.*, 52(6):1558–1567, 2003.

[Cla05] H. Claussen. Efficient modelling of channel maps with correlated shadow fading in mobile radio systems. *Proc. IEEE PIMRC' 05*, 1:512 –516, 2005.

[EHB+13] Ernst Eberlein, Thomas Heyn, Frank Burkhardt, Stephan Jaeckel, Lars Thiele, Thomas Haustein, Gerd Sommerkorn, Martin Käske, Christian Schneider, Maria Dominguez, and Joel Grotz. Characterisation of the MIMO channel for mobile satellite systems (acronym: MIMOSA), TN8.2 – final report. Technical Report v1.0, Fraunhofer Institute for Integrated Circuits (IIS), 2013.

[Gud91] M. Gudmundson. Correlation model for shadow fading in mobile radio systems. *IET Electron Lett.*, 27(23):2145–2146, November 1991.

[Hat80] M. Hata. Empirical formula for propagation loss in land mobile radio services. *IEEE Trans. Veh. Technol.*, 29(3):317–325, 1980.

[HMK+10] Petteri Heino, Juha Meinilä, Pekka Kyösti, et al. CELTIC / CP5-026 D5.3: WINNER+ final channel models. Technical report, 2010.

[JRB+14] S. Jaeckel, L. Raschkowski, K. Börner, L. Thiele, F. Burkhardt, and E. Eberlein. QuaDRiGa - Quasi Deterministic Radio Channel Generator, User Manual and Documentation. Technical Report v1.1.0-248, Fraunhofer Heinrich Hertz Institute, 2014.

[KMH+07] Pekka Kyösti, Juha Meinilä, Lassi Hentilä, et al. IST-4-027756 WINNER II D1.1.2 v.1.1: WINNER II channel models. Technical report, 2007.

[MTIO09] L. Materum, J. Takada, I. Ida, and Y. Oishi. Mobile station spatio-temporal multipath clustering of an estimated wideband MIMO double-directional channel of a small urban 4.5 GHz macrocell. *EURASIP J. Wireless Commun. Netw.*, 2009.

[NKS+07] M. Narandzic, M. Käske, C. Schneider, M. Milojevic, M. Landmann, G. Sommerkorn, and R.S. Thomä. 3D-antenna array model for IST-WINNER channel simulations. *Proc. IEEE VTC '07 Spring*, pages 319–323, 2007.

[NSK+11] M. Narandzic, C. Schneider, M. Käske, S. Jaeckel, G. Sommerkorn, and R.S. Thomä. Large-scale parameters of wideband MIMO channel in urban multi-cell scenario. *Proc. EUCAP '11*, 2011.

[OCGD08] C. Oestges, B. Clerckx, M. Guillaud, and M. Debbah. Dual-polarized wireless communications: From propagation models to system performance evaluation. *IEEE Trans. Wireless Commun.*, 7(10):4019–4031, 2008.

[PB01] M.F. Pop and N.C. Beaulieu. Limitations of sum-of-sinusoids fading channel simulators. *IEEE Trans. Commun.*, 49(4):699–708, 2001.

eMail: quadriga@hhi.fraunhofer.de

[PHL+11] J. Poutanen, K. Haneda, Lingfeng Liu, C. Oestges, F. Tufvesson, and P. Vainikainen. Parameterization of the COST 2100 MIMO channel model in indoor scenarios. *Proc. EUCAP '11*, pages 3606–3610, 2011.

[PMF97] K.I. Pedersen, P.E. Mogensen, and B.H. Fleury. Power azimuth spectrum in outdoor environments. *Electronics Letters*, 33(18):1583–1584, 1997.

[QOHDD10] F. Quitin, C. Oestges, F. Horlin, and P. De Doncker. A polarized clustered channel model for indoor multiantenna systems at 3.6 GHz. *IEEE Trans. Veh. Technol.*, 59(8):3685–3693, 2010.

[Rap02] T.S. Rappaport. *Wireless Communications. Principles and Practice.* Prentice Hall, 2 edition, 2002.

[SNK+10] C. Schneider, M. Narandzic, M. Käske, G. Sommerkorn, and R.S. Thomä. Large scale parameter for the WINNER II channel model at 2.53 GHz in urban macro cell. *Proc. IEEE VTC '10 Spring*, 2010.

[Sva01] T. Svantesson. A physical MIMO radio channel model for multi-element multi-polarized antenna systems. *Proc. IEEE VTC' 01 Fall*, 2:1083–1087, 2001.

[SZM+06] M. Shafi, Min Zhang, A.L. Moustakas, P.J. Smith, A.F. Molisch, F. Tufvesson, and S.H. Simon. Polarized MIMO channels in 3-D: models, measurements and mutual information. *IEEE J. Sel. Areas Commun.*, 24:514–527, Mar. 2006.

[ZRP+05] Y. Zhou, S. Rondineau, D. Popovic, A. Sayeed, and Z. Popovic. Virtual channel space-time processing with dual-polarization discrete lens antenna arrays. *IEEE Trans. Antennas Propag.*, 53:2444–2455, Aug. 2005.

eMail: quadriga@hhi.fraunhofer.de

1 Introduction and Overview

1.1 Installation and System Requirements

The installation is straightforward and it does not require any changes to your system settings. If you would like to use QuaDRiGa, just extract the ZIP-File containing the model files and add the "source"-folder from the extracted archive to you MATLAB-Path. This can be done by opening MATLAB and selecting "File" - "Set Path ..." from the menu. The you can use the "Add folder ..." button to add QuaDRiGa to your MATLAB-Path.

Table 1: QuaDRiGa System Requirements

Requirement	Value
Minimal required MATLAB version	7.12 (R2011a)
Required toolboxes	none
Memory (RAM) requirement	1 GB
Processing power	1 GHz Single Core
Storage	50 MB
Operating System	Linux, Windows, Mac OS

1.2 General Remarks

This document gives a detailed overview of the `QuaDRiGa` channel model and its implementation details. The model has been evolved from the Wireless World Initiative for New Radio (WINNER) channel model described in WINNER II deliverable D1.1.2 v.1.1 [KMH+07]. This document covers only the model itself. Measurement campaigns covering the extraction of suitable parameters can be found in the WINNER documentation [KMH+07, HMK+10] or other publications such as [SNK+10, NSK+11]. Furthermore, the MIMOSA project [EHB+13] covers the model development and parameter extraction for land-mobile satellite channels.

The `QuaDRiGa` channel model follows a geometry-based stochastic channel modeling approach, which allows the creation of an arbitrary double directional radio channel. The channel model is antenna independent, i.e. different antenna configurations and different element patterns can be inserted. The channel parameters are determined stochastically, based on statistical distributions extracted from channel measurements. The distributions are defined for, e.g. delay spread, delay values, angle spread, shadow fading, and cross-polarization ratio. For each channel segment the channel parameters are calculated from the distributions. Specific channel realizations are generated by summing contributions of rays with specific channel parameters like delay, power, angle-of-arrival and angle-of- departure. Different scenarios are modeled by using the same approach, but different parameters. The basic features of the model approach can be summarized as follows:

- Support of freely configurable network layouts with multiple transmitters and receivers
- Scalability from a single input single output (SISO) or multiple-input multiple-output (MIMO) link to a multi-link MIMO scenario
- Same modeling approach indoor, outdoor, and satellite environments as well as combinations of them
- Support of a frequency range of 2-6 GHz with up to 100 MHz RF bandwidth
- Support of multi-antenna technologies, polarization, multi-user, multi-cell, and multi-hop networks
- Smooth time evolution of large-scale and small-scale channel parameters including the transition between different scenarios
- High accuracy for the calculation of the polarization characteristics

The QuaDRiGa channel model largely extends the WINNER model to support several new features that were originally not included. These are

- Time evolution
  Short term time evolution of the channel coefficients is realized by updating the delays, the departure- and arrival angels, the polarization, the shadow fading and the K-Factor based on the position of the terminal.
- Scenario transitions
  When the mobile terminal (MT) moves through the fading channel, it may pass through several different scenarios. QuaDRiGa supports smooth transitions between adjacent channel segments. This is used to emulate long term time evolution and allows the simulation of e.g. handover scenarios.
- Variable speeds for mobile terminals
  QuaDRiGa supports variable speeds including accelerating and sowing down of mobile terminals.
- Common framework for LOS and NLOS simulations
  In WINNER, line of sight (LOS) and non line of sight (NLOS) scenarios were treated differently. QuaDRiGa used the same method for both scenarios types. This reduces the model complexity and enables freely configurable multicell scenarios. E.g. one MT can see two base stations (BSs), one in LOS and another in NLOS.
- Geometric polarization
  The polarizations for the LOS and for the NLOS case is now calculated based on a ray-geometric approach.
- Improved method for calculating correlated large scale parameters (LSPs)
  The WINNER model calculates maps of correlated parameter values using filtered random numbers. QuaDRiGa uses the same method but extends the map generation algorithm to also consider diagonal movement directions and to create smoother outputs.
- New functions for modifying antenna patterns
  Antenna patterns can now be freely rotated in 3D-coordinates while maintaining the polarization properties.
- New MATLAB implementation
  The MATLAB code was completely rewritten. The implementations now fosters object oriented programming and object handles. This increases the performance significantly and lowers the memory usage.

1.3 Introduction to QuaDRiGa

QuaDRiGa (QUAsi Deterministic RadIo channel GenerAtor) was developed to enable the modeling of MIMO radio channels for specific network configurations, such as indoor, satellite or heterogeneous configurations.

Besides being a fully-fledged three dimensional geometry-based stochastic channel model, QuaDRiGa contains a collection of features created in spatial channel model (SCM) and WINNER channel models along with novel modeling approaches which provide features to enable quasi-deterministic multi-link tracking of users (receiver) movements in changing environments.

The main features of QuaDRiGa are:

- Three dimensional propagation (antenna modeling, geometric polarization, scattering clusters),
- Continuous time evolution,
- Spatially correlated propagation parameter maps,
- Transitions between varying propagation scenarios

The QuaDRiGa approach can be understood as a "statistical ray-tracing model". Unlike the classical ray tracing approach, it doesn't use an exact geometric representation of the environment but distributes the positions of the scattering clusters (the sources of indirect signals such as buildings or trees) randomly. A simplified overview of the model is depicted in Figure 2. For each path, the model derives the angle of departure (the angle between the transmitter and the scattering cluster), the angle of arrival (the angle between the receiver and the scattering cluster) and the total path length which results in a delay τ of the signal. For the sake of simplicity, only two paths are shown in the figure.

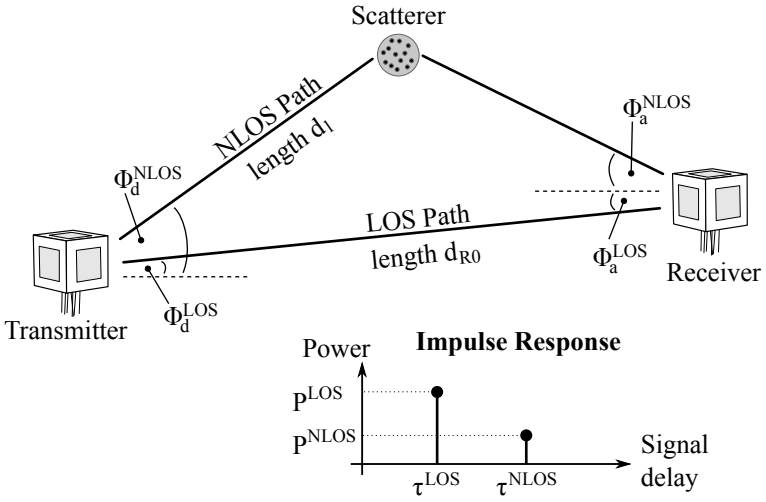

Figure 1: Simplified overview of the modeling approach used in `QuaDRiGa`

Terrestrial and Satellite scenarios can be modeled. For "Satellite to Earth" communication the angle of departure is identical for all clusters. The concept behind the model allows also the modeling of scenarios such as

- Earth to satellite
- Satellite systems with complementary ground components (CGC): Using several transmitters at different positions and simulating all propagation paths in one setup is supported.

The analysis of these scenarios was not in the scope of the MIMO over Satellite (MIMOSA) project. This feature is not tested and especially no parameter sets are available yet.

In the following, the terms cluster, scattering cluster and scatterer are used synonymously. A cluster describes an area where many scattering events occur simultaneously, e.g. at the foliage of trees or at a rough building wall. In QuaDRiGa, each scattering cluster is approximated by 20 individual scatterers. Each one is modeled by a single reflection. The 20 signals can be resolved in spatial domain where they have a typical angular spread of 1-6°. However, they cannot be resolved in delay domain. Therefore, in the output of the channel model, these 20 signals (also named sub-paths) are combined into a single signal which is represented by a path. The difference to Rayleigh fading models, which use wide sense stationary uncorrelated scattering (WSSUS) taps instead of paths, is that each path has a very limited angular spread (1-6°) which also results in a narrow Doppler spectrum. The terms path, multipath component (MPC) and tap are also used synonymously in the QuaDRiGa documentation.

To emulate a rich scatting environment with a wider angular spread, many scattering clusters are created. QuaDRiGa supports up to 42 clusters. However, depending on the angular spread and the amount of diffuse scatting (which is approximated by discrete clusters in QuaDRiGa), typical values are around 10 cluster for LOS propagation and 20 clusters for non-LOS. The positioning of the clusters is controlled by the environment angular spread and the delay spread. The environment angular spread has values of around 20-90° and is typically much larger than the per-cluster angular spread. However, even with many clusters, the Doppler spread is narrower in QuaDRiGa than when assuming pure Rayleigh fading. This is also in line

with measurement results. It can be observed in the field that the main components arrive from selected angles and the classical Doppler spectrum's "Jakes" or Butterworth filter shaped characteristics are only valid as long term average and not valid for a short time interval.

To summarize:

- A typical propagation environment requires 8-20 clusters.
- Internally, each cluster is represented by 20 sub-paths, resulting in 160 - 400 sub-paths in total.
- Each sub-path is modeled as a single reflection.
- The 160 - 400 sub-paths are weighted by the antenna response. The 20 sub-paths for each cluster are summed up which results in 8-20 paths.
- For a MIMO system with multiple antennas at the transmitter and receiver, each path has as many channel coefficients, as there are antenna pairs. Hence, at the output, there are $n_{Path} \cdot n_{Rx} \cdot n_{Tx}$ channel coefficients.

1.4 Continuous time evolution

QuaDRiGa calculates the channel for each defined reception point. To generate a "time series" a continuous track of reception points can be defined. The arrival angles of the sub-paths play a crucial for the time evolution because the phase changes are calculated deterministically based on the arrival angles. This results in a realistic Doppler spectrum.

The temporal evolution of the channel is modeled by two effects:

- drifting and
- birth and death of clusters.

Drifting (see Section 3.2.3) occurs within a small area (about 20-30 m diameter) in which a specific cluster can be seen from the MT. Within this area the cluster position is fixed. Due to the mobility of the terminal the path length (resulting in a path delay) and arrival angels change slowly.

Longer time-evolving channel sequences need to consider the birth and death of scattering clusters as well as transitions between different propagation environments. We address this by splitting the MT trajectory into segments. A segment can be seen as an interval in which the LSPs, e.g. the delay and angular spread, do not change considerably and where the channel keeps its wide sense stationary (WSS) properties. Thus, the length of a segment depends on the decorrelation distances of the LSPs. We propose to limit the segment length to the average decorrelation distance. Typical values are around 20 m for LOS and 45 m for NLOS propagation. In the case where a state does not change over a long time, adjacent segment must have the same state. For example, a 200 m NLOS segment should be split into at least 4 NLOS sub-segments.

A set of clusters is generated independently for each segment. However, since the propagation channel does not change significantly from segment to segment, we need to include correlation. This is done by so called "parameter maps" (see Section 3.1.4). The maps ensure that neighboring segments do not have significantly different propagation characteristics. For example, measurements show that the shadow fading (the average signal attenuation due to building, trees, etc.) is correlated over up to 100 m. Hence, we call all channel characteristics showing similarly slow changes LSPs.

With a segment length of 20 m, two neighboring segments of the same state will have similar receive-power. To get the correct correlation, QuaDRiGa calculates a map for the average received power for a large area. The received power for two adjacent segments is then obtained by reading the values of the map. This map-based approach also contains cross-correlations to other LSPs such as the delay spread. For example, a shorter delay spread might result in a higher received power. Hence, there is a positive correlation between power and delays spread which is also reflected in the maps.

To get a continuous time-series of channel coefficients requires that the paths from different segments are combined at the output of the model. In between two segments clusters from the old segment disappear

eMail: quadriga@hhi.fraunhofer.de

and new cluster appear. This is modeled by merging the channel coefficients of adjacent segments. The active time of a scattering cluster is confined within the combined length of two adjacent segments. The power of clusters from the old segment is ramped down and the power of new clusters is ramped up within the overlapping region of the two segments. The combination clusters to ramp up and down is modeled by a statistical process. Due to this approach, there are no sudden changes in the LSPs. For example, if the delay spread in the first segment is 400 ns and in the second it is 200 ns, then in the overlapping region, the delay spread (DS) slowly decreases till it reaches 200 ns. However, this requires a careful setup of the segments along the used trajectory. If the segments are too short, sudden changes cannot be excluded. This process is described in detail in Section 3.2.7.

1.5 QuaDRiGa Program Flow

For a propagation environment (e.g. urban, suburban, rural or tree-shadowing) typical channel characteristics are described by statistics of the LSPs. Those are the median and the standard deviation of the delay spread, angular spreads, shadow fading, Ricean K-Factor, as well as correlations between them. Additional parameters describe how fast certain properties of the channel change (i.e. the decorrelation distance). Those parameters are stored in configuration files which can be edited by the model user. Normally, the parameters are extracted from channel measurements. A detailed description of the model steps can be found Section 3.

1. The user of the model needs to configure the network layout. This includes:

   - Setting the transmitter position (e.g. the BS positions or the satellite orbital position)
   - Defining antenna properties for the transmitter and the receiver
   - Defining the user trajectory
   - Defining states (or segments) along the user trajectory
   - Assigning a propagation environment to each state

   Defining the user trajectory, states along the user trajectory and related parameters is performed by the state sequence generator (SSG). In the current implementation different SSGs are available:

   - Manual definition of all parameters by the user, e.g. definition of short tracks.
   - Statistical model for the "journey". A simple model (mainly designed for demonstration and testing purpose is included in the tutorial "satellite_channel")
   - Derive trajectory and state sequence from the measurement data.

2. Configuration files define the statistical properties of the LSPs. For each state (also called scenario) a set of properties is provided. Typically two configurations files are used.

   - One for the "good state" (also called LOS scenario)
   - The other for the "bad state" (NLOS scenario).

   For each state QuaDRiGa generates correlated "maps" for each LSP. For example, the delay spread in the file is defined as log-normal distributed with a range from 40 to 400 ns. QuaDRiGa translates this distribution in to a series of discrete values, e.g. 307 ns for segment 1, 152 ns for segment 2, 233 ns for segment 3 and so on. This is done for all LSPs.

3. The trajectory describes the position of the MT in the "maps". For each segment of the trajectory, clusters are calculated according to the values of the LSPs at the map position. The cluster positions are random within the limits given by the LSP. For example, a delay spread of 152 ns limits the distance between the clusters and the terminal.

4. Each cluster is split into 20 sub-paths and the arrival angles are calculated for each sub-path and for each positions of the terminal on the trajectory.

5. The antenna response for each of the arrival angels is calculated (the same holds for the departure angles). If there is more than one antenna at the transmitter- and/or receiver side, the calculation is repeated for each antenna.

6. The phases are calculated based on the position of the terminal antennas in relation to the clusters. The terminal trajectory defines how the phases change. This results in the Doppler spread.

7. The coefficients of the 20 sup-paths are summed (the output are paths). If there is more than one antenna and depending on the phase, this sum results in a different received power for each antenna-pair. At this point, the MIMO channel response is created.

8. The channel coefficients of adjacent segments are combined (merged). This includes the birth/death process of clusters. Additionally, different speeds of the terminal can be emulated by interpolation of the channel coefficients.

9. The channel coefficients together with the path delays are formatted and returned to the user for further analysis.

1.6 Description of modeling of different reception conditions by means of a typical drive course

This section describes some of the Key features of the model using a real world example. A detailed introduction with a variety of tutorials, test cases and interface descriptions then follows in section A. The later part of the document then focusses on the mathematical models behind the software and the assumptions made.

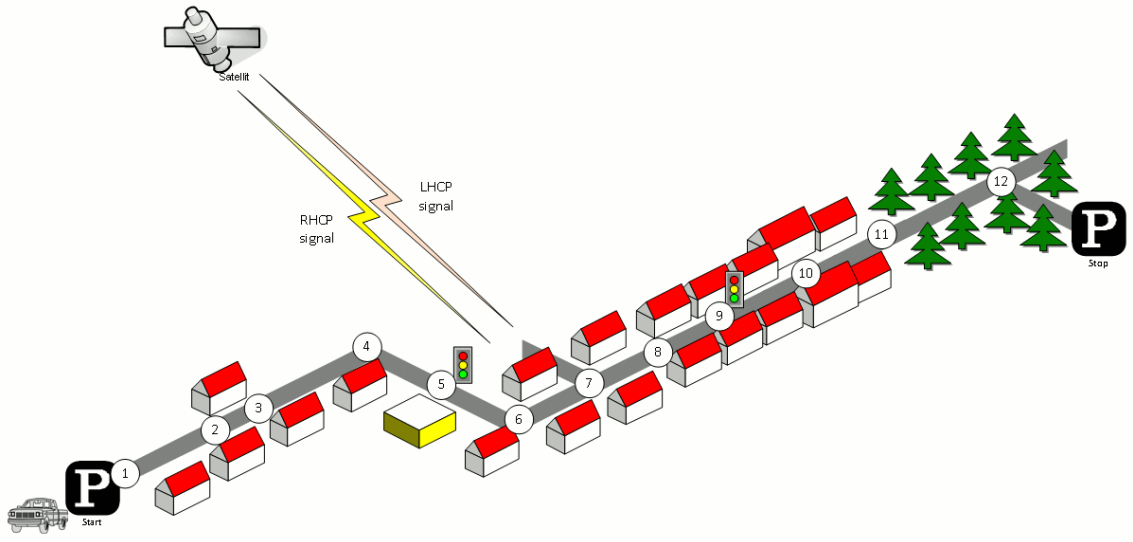

Figure 2: Typical driving course: From home to woodland parking site on the village outskirts

The different effects along the track can be summarized as follows:

1. Start Environment: Urban, LOS reception of satellite signal
2. LOS → NLOS Change
3. NLOS → LOS Change
4. Turning off without change in reception condition (LOS)
5. Stopping at traffic light (LOS)
6. Turning off with change of reception condition (LOS → NLOS)
7. Crossing side Street (NLOS → short LOS → NLOS)

8. Structural change in the environment without a change in the environment type (higher density of buildings but still the environment remains urban)
9. Stopping at traffic lights (NLOS)
10. Houses have the same characteristics as before but are further away from the street (urban environment with different reception characteristics)
11. Change of environment (Urban → Forest)
12. Turning off without change of environment (NLOS)

Each simulation run in QuaDRiGa is done in three (and an optional fourth) step.

1. Set up tracks, scenarios, antennas and network layout
2. Generate correlated LSPs
3. Calculate the channel coefficients
4. (optional) Post-processing

Those steps also need to be done for the above scenario. However, different aspects of the track are handled in different parts of the model. Additionally, the QuaDRiGa model supports two operating modes for handling the LSPs:

1. The first (default) mode generates the correlated LSPs automatically based on a scenario-specific parameter set. This is done in step 2 and involves so called parameter maps.

2. The manual mode does not generate LSPs automatically. Here, the user has to supply a list of parameters to the model. The step 2 thus to be implemented by the user.

Steps 1, 3 and 4 are identical for both modes. The following list describes the modeling of the observed effects along the track when using the automatic mode (1).

1. **Start Environment: Urban, LOS reception of satellite signal**
   Each segment along the track gets assigned an environment. In the QuaDRiGa terminology, this is called a scenario. E.g. the first segment on the track is in the "Satellite-LOS-Urban"-scenario. The selection of the scenario is done during the first step (set up tracks, scenarios, antennas and network layout). QuaDRiGa itself does not supply functions to perform the setting up of tracks and scenarios automatically. However, external scripts can be used to perform this task. An example can be found in section A.3. A RHCP/LHCP signal is defined in the antenna setup.

   After the model setup, the "automatic mode" generates a set of LSPs for this segment. I.e. the second step of the model calculates one value for each of the 7 LSPs using the map-based method. Thus, a set of seven maps is created for the scenario "Satellite-LOS-Urban". Those maps cover the entire track. Thus, the same maps are used for all "Satellite-LOS-Urban"-segments of the track.

   The third step then calculates a time-series of fading coefficients for the first segment that have the properties of the LSPs from the map. E.g. if one calculates the RMS-DS from the coefficients, one gets the same value as generated by the map in step 2.

2. **LOS → NLOS Change**
   A scenario change is defined along the track. E.g. the second segment along the track gets assigned the scenario "Satellite-NLOS-Urban". Now, a second set of maps is generated for all "Satellite-NLOS-Urban"-segments. So in total, we now have 14 maps (seven for LOS and another seven for NLOS). The parameters for calculating the channel coefficients are drawn from the second seven maps.

   We get a set of channel coefficients with different properties (e.g. more multipath components, lower K-Factor etc.). A smooth transition between the coefficients from the first segment and the second is realized by the ramping down the powers of the clusters of the old segment and ramping up the power of the new. This is implemented in step 4 (Post-processing).

3. **NLOS → LOS Change**
   This is essentially the same as in point 2. However, since the third segment is also in the scenario "Satellite-LOS-Urban", no new maps are generated. The parameters are extracted from the same map as for the starting segment.

4. **Turning off without change in reception condition (LOS)**
   QuaDRiGa supports free 3D-trajectories for the receiver. Thus, no new segment is needed - the terminal stays in the same segment as in point 3. However, we assume that the receive antenna is fixed to the terminal. Thus, if the car turns around, so does the antenna. Hence, the arrival angles of all clusters, including the direct path, change. This is modeled by a time-continuous update of the angles, delays and phases of each multipath component, also known as drifting. Due the change of the arrival angles and the path-lengths, the terminal will also see a change in its Doppler-profile.

5. **Stopping at traffic light (LOS)**
   QuaDRiGa performs all internal calculations at a constant speed. However, a stop of the car at a traffic light is realized by interpolating the channel coefficients in an additional post processing step (step 4). Here, the user needs to supply a movement profile that defines all acceleration, deceleration or stopping points along the track. An example is given in section A.6. Since the interpolation is an independent step, it makes no difference if the mobile terminal is in LOS or NLOS conditions.

6. **Turning off with change of reception condition (LOS → NLOS)**
   This is realized by combining the methods of point 2 (scenario change) and point 4 (turning without change). The scenario change is directly in the curve. Thus, the LOS and the NLOS segments have an overlapping part where the cluster powers of the LOS segment ramp down and the NLOS clusters ramp up. The update of the angles, delays and phases is done for both segments in parallel.

7. **Crossing side Street (NLOS → short LOS → NLOS)**
   This is modeled by two successive scenario changes (NLOS-LOS and LOS-NLOS). For both changes, a new set of clusters is generated. However, since the parameters for the two NLOS-segments are extracted from the same map, they will be highly correlated. Thus, the two NLOS segments will have similar properties.

8. **Structural change in the environment without a change in the environment type** (higher density of buildings but still the environment remains urban)
   This is not explicitly modeled. However, the "Satellite-NLOS-Urban"-map covers a typical range of parameters. E.g. in a light NLOS area, the received power can be some dB higher compared to an area with denser buildings. The placement of light/dense areas on the map is random. Thus, different characteristics of the same scenario are modeled implicit. They are covered by the model, but the user has no influence on where specific characteristics occur on the map when using the automatic mode. An alternative would be to manually overwrite the automatically generated parameters or use the manual mode.

   In order to update the LSPs and use a new set of parameters, a new segment needs to be created. I.e. here, an environment change from "Satellite-NLOS-Urban" to the same "Satellite-NLOS-Urban" has to be created. Thus, a new set of LSPs is read from the map and new clusters are generated accordingly.

9. **Stopping at traffic lights (NLOS)**
   This is the same as in point 5.

10. **Structural change in environment**. Houses have the same characteristics as before but are further away from the street (urban environment with different reception characteristics)
    Same as point 8.

11. **Change of environment (Urban → Forest)**
    This is the same as in point 2. The segment on the track gets assigned the scenario "Satellite-Forest" and a third set of maps (15-21) is generated for the "Satellite-Forest"-segment. The parameters are drawn from those maps, new channel coefficients are calculated and the powers of the clusters are ramped up/down.

12. **Turning off without change of environment (NLOS)**
    Same as in point 4.

2 Software Structure

2.1 Overview

QuaDRiGa is implemented in MATLAB using an object oriented framework. The user interface is built upon classes which can be manipulated by the user. Each class contains fields to store data and methods to manipulate the data. An overview of the class structure is given in Section 2.

It is important to keep in mind that all classes in QuaDRiGa are "handle"-classes. This significantly reduces memory usage and speeds up the calculations. However, **all MATLAB variable names assigned to QuaDRiGa objects are pointers**. If you copy a variable (i.e. by assigning "**b = a**"), only the pointer is copied. "**a**" and "**b**" point to the same object in memory. If you change the values of "**b**", the value of "**a**" is changed as well. This is somewhat different to the typical MATLAB behavior and might cause errors if not considered properly. Copying a QuaDRiGa object can be done by "**b = a.copy**".

- **User input**
  The user inputs (Point 1 in the programm flow) are provided through the classes: "simulation_parameters", "array", "track", and "layout".

  "**simulation_parameters**" defines the general settings such as the center frequency and the sample density. It also enables and disables certain features of the model such as polarization rotation, subpath output and progress bars.

  "**array**" combines all functions needed to describe antenna arrays.

  "**track**" is used to define user trajectories, states and segments.

  "**layout**" is a object including the tracks and antenna properties together with further parameters such as the satellite position.

- **Internal processing**
  All the processing is done by the classes "parameter_set" and "channel_builder".

  "**parameter_set**" is responsible for generating LSPs for the cluster generation. It also holds the parameter maps needed for generating auto- and crosscorrelation properties of the parameters. "parameter_set" implements point 2 of the program flow.

  "**channel_builder**" creates the channel coefficients. This includes the cluster generation and the MIMO channels. It implements steps 3-7 of the program flow.

- **Model output**
  The final two steps (8 and 9) of the program flow are implemented in the class "**channel**". Objects of this class hold the data for the channel coefficients. The class also implements the channel merger, which creates long time evolving sequences out if the snipes produced by the channel builder. Additional function such as the transformation into frequency domain can help the user to further process the data.

An overview of the model software is depicted in Fig. 3. The unified modeling language (UML) class diagram of the `QuaDRiGa` channel model gives an overview of all the classes, methods and properties of the model. The class diagram serves as a reference for the following descriptions which also lists the methods that implement a specific functionality.

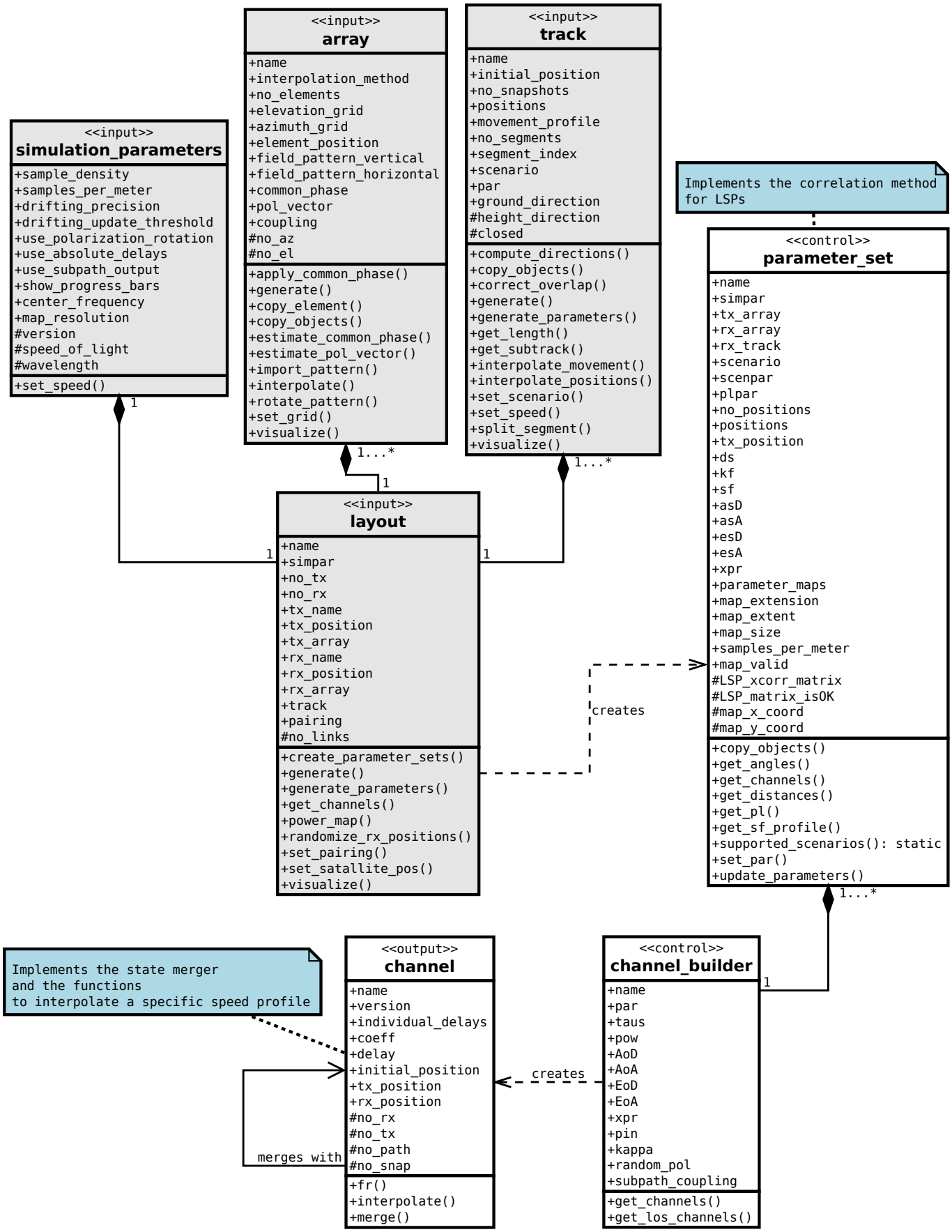

Figure 3: UML class diagram of the model software.

2.2 Description of Classes, Properties, and Methods

In the following, all properties and methods of the QuaDRiGa classes are described. For the methods, input and output variables are defined and explained. There are three types of methods: Standard methods require an instance of a class. They are printed in black without the class name:

par = **generate_parameters** (overlap, usage, check_parfiles, verbose)

Static methods can be called directly from the command line without creating an instance of the class first. They are printed in blue:

[h_array, mse, mse_pat] = **array.import_pattern** (fVi, fHi , correct_phase, accuracy, max_num_elements, azimuth_grid, elevation_grid, verbose)

The constructor is a special method that is called when the class name is used as a function, e.g. when calling `a = array('dipole')`. There is only one constructor for each class. They are printed in blue.

h_array = **array** (array_type, phi_3dB, theta_3dB, rear_gain)

Index of Methods

  - generate (static)
  - generate_parameters
  - get_channels
  - power_map
  - randomize_rx_positions
  - set_pairing
  - set_satellite_pos
  - visualize

- Class "parameter_set"

  - parameter_set (constructor)
  - copy_objects
  - get_angles
  - get_channels
  - get_distances
  - get_pl
  - get_sf_profile
  - set_par
  - supported_scenarios (static)
  - update_parameters

- Class "channel_builder"

  - channel_builder (constructor)
  - get_channels
  - get_los_channels (static)

- Class "channel"

  - channel (constructor)
  - fr
  - interpolate
  - merge

2.2.1 Class "simulation_parameters"

This class controls the simulation options and calculates constants for other classes.

Properties

sample_density	The number of samples per half-wave length Sampling density describes the number of samples per half-wave length. To fulfill the sampling theorem, the minimum sample density must be 2. For smaller values, interpolation of the channel for variable speed is not possible. On the other hand, high values significantly increase the computing time significantly. A good value is around 4.
samples_per_meter	Samples per meter This parameter is linked to the sample density by $$f_S = 2 \cdot f_C \cdot \frac{\mathrm{SD}}{c}$$ where f_C is the carrier frequency in Hz, SD is the sample density and c is the speed of light.
drifting_precision	Precision of the drifting functionality drifting_precision = 0 This method applies rotating phasors to each path which emulates time varying Doppler characteristics. However, the large-scale parameters (departure and arrival angles, shadow fading, delays, etc.) are not updated in this case. This mode requires the least computing resources and may be preferred when only short linear tracks (up to several cm) are considered and the distance between transmitter and receiver is large. drifting_precision = 1 (default) When drifting is enabled, all LSPs are updated for each snapshot. This requires significantly more computing resources but also increases the accuracy of the results. Drifting is required when e.g. non-linear tracks are generated or the distance between transmitter and receiver is small (below 20 m). drifting_precision = 2 LSPs are updated for each snapshot and for each antenna element at the receiver. This increases the accuracy for multi-element antenna arrays. drifting_precision = 3 This option also calculates the shadow fading, path loss and K-factor for each antenna separately. This feature tends to predict higher capacities since is also increases the randomness of the power for different MIMO elements. Use with care.
drifting_update_threshold	Update drifting paths when arrival angle changes [degrees] Updating the polarization rotation for large antenna arrays is very computing intensive and may slow down the simulations significantly. This parameter reduces the update rate. A value of 0.2° (Default Setting) means that the polarization rotation is only updated when any of the four angles (AoA, EoA, AoD, EoD) changes more than 0.2°. Otherwise it is kept constant at its last value. For highest precision calculations, set '`drifting_update_threshold`' to 0.
use_polarization_rotation	Select the polarization rotation method use_polarization_rotation = 0 Uses the polarization method from WINNER. No polarization rotation is calculated. use_polarization_rotation = 1 Uses the new polarization rotation method where the cross polarization ratio (XPR) is modeled by a rotation matrix. No change of circular polarization is assumed. use_polarization_rotation = 2 [Default] Uses the polarization rotation with an additional phase offset between the H and V component of the NLOS paths. The offset angle is calculated to match the XPR for circular polarization. use_polarization_rotation = 3 Uses polarization rotation for the geometric polarization but models the NLOS polarization change as in WINNER.
use_absolute_delays	Returns absolute delays in channel impulse response (CIR). By default, delays are calculated such that the LOS delay is normalized to 0. By setting '`use_absolute_delays`' to 1 or '`true`', the absolute path delays are included in '`channel.delays`' at the output of the model.

use_subpath_output	Return all 20 subpaths By default, the complex valued amplitudes of 20 sub-paths are summed up to calculate the path amplitude. Setting '`use_subpath_output`' to 1 returns the individual amplitudes of each subpath in '`channel.coeff`' at the output of the model.
show_progress_bars	Show a progress bar on the MATLAB prompt
center_frequency	Center frequency in [Hz]
map_resolution	Resolution of the decorrelation maps in [samples/m]
version	Version number of the current QuaDRiGa release (constant)
speed_of_light	Speed of light (constant)
wavelength	Carrier wavelength in [m] (read only)

Methods

h_simpar = **simulation_parameters**

Description	Creates a new '`simulation_parameters`' object with default settings.

set_speed (speed_kmh, sampling_rate_s)

Description	This method can be used to automatically calculate the sample density for a given mobile speed.	
Input	speed_kmh	speed in [km/h]
	sampling_rate_s	channel update rate in [s]

2.2.2 Class "array"

This class combines all functions to create and edit antenna arrays. An antenna array is a set of single antenna elements, each having a specific beam pattern, that can be combined in any geometric arrangement. A set of synthetic arrays that allow simulations without providing your own antenna patterns is provided (see generate method for more details).

Properties

name	Name of the antenna array
interpolation_ method	Method for interpolating the beam patterns The default is linear interpolation. Optional are: • nearest - Nearest neighbor interpolation (QuaDRiGa optimized) • **linear** - Linear interpolation (QuaDRiGa optimized, Default) • spline - Cubic spline interpolation (MATLAB internal function) • nearest_int - Nearest neighbor interpolation (MATLAB internal function) • linear_int - Linear interpolation (MATLAB internal function) Note: MATLAB internal routines slow down the simulations significantly.
no_elements	Number of antenna elements in the array Increasing the number of elements creates new elements which are initialized as copies of the first element. Decreasing the number of elements deletes the last elements from the array.
elevation_grid	Elevation angles in [rad] were samples of the field patterns are provided The field patterns are given in spherical coordinates. This variable provides the elevation sampling angles in radians ranging from $-\frac{\pi}{2}$ (downwards) to $\frac{\pi}{2}$ (upwards).
azimuth_grid	Azimuth angles in [rad] were samples of the field patterns are provided The field patterns are given in spherical coordinates. This variable provides the azimuth sampling angles in radians ranging from $-\pi$ to π.
element_position	Position of the antenna elements in local cartesian coordinates (using units of [m])
field_pattern_ vertical	Vertical (or theta) component of the electric field given in spherical coordinates. This variable is a tensor with dimensions [elevation, azimuth, element] describing the vertical (or theta) component of the far field of each antenna element in the array.
field_pattern_ horizontal	Horizontal (or phi) component of the electric field given in spherical coordinates. This variable is a tensor with dimensions [elevation, azimuth, element] describing the horizontal (or phi) component of the far field of each antenna element in the array.
common_phase	A phase offset which is the same on both polarizations Internally, QuaDRiGa can only use real-valued patterns. Complex valued patterns have to be split into at least two real valued patterns. However, if the phase is the same on both polarization components, i.e. the antenna is linearly polarized but has a direction-dependant phase, then the phase might be stored here. This variable is a tensor with dimensions [elevation, azimuth, element]. By default, it is initialized with zeros (no phase offset).
pol_vector	The orientation of the electric field in 3D local cartesian coordinates.
coupling	Coupling matrix between elements This matrix describes a pre or postprocessing of the signals that are fed to the antenna elements. For example, in order to transmit a left hand circular polarized (LHCP) signal, two antenna elements are needed. They are then coupled by a matrix $\frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ j \end{pmatrix}$ The rows in the matrix correspond to the antenna elements, the columns to the signal ports. In this example, the antenna has one port, i.e. it is fed with one input signal. This signal is then split into two and fed to the two antenna elements where the second element radiates the signal with 90° phase shift. In a similiar fasion, it is possible to create fixed beamforming antennas and include crosstalk between antenna elements. By default, '`coupling`' is set to an identity matrix which indicates perfect isolation between the antenna elements.
no_az	Number of azimuth values
no_el	Number of elevation values

Methods

h_array = **array** (array_type, phi_3dB, theta_3dB, rear_gain)	
Description	Creates a new array object. See 'array.generate' for a description of the input parameters and the list of supported antenna types.

apply_common_phase (element)		
Description	Applies the common phase to the patterns.	
Input	element	The element numbers for which this functions is applied. If no element number is given, the common phase is applied to all elements in the array.

copy_element (source, target)		
Description	Creates a copy of an antenna element.	
Input	source	Index of the array object that should be copied. The value must be scalar, integer and greater than 0 and it can not exceed the array size.
	target	Target can be a scalar or vector with elements > 0.

copy_objects	
Description	A modified version of the standard physical copy function While the standard copy command creates new physical objects for each element of obj (in case obj is an array of object handles), copy_objects checks whether there are object handles pointing to the same object and keeps this information.

estimate_common_phase (element)		
Description	Estimates the common phase from the field patterns It is possible that antenna patterns have a phase component. For example, an antenna array might be assembled out of several elements, which are at different position in the array. When the patterns of the elements are then calibrated in the lab, the individual positions result in a phase offset which is part of the measured pattern response. The core function of QuaDRiGa, however, only uses the real part if the provided patterns. Hence, not calibrating the phase offset out of the pattern will lead to errors. This function calculates the phase from the pattern and stores it in the '`common_phase`' property of the antenna array object. However, this requires that the orientation vectors are set correctly, i.e. you need to call '`estimate_pol_vector`' first when importing measured patterns.	
Input	element	The element numbers for which this functions is applied. If no element number is given, the common phase is estimated for all elements in the array.

[pol_vector, err] = **estimate_pol_vector** (element, verbose)		
Description	Estimates the orientation vector from the patterns This function estimates the orientation vector from the field patterns. This allows the use of measured patterns in QuaDRiGa where the orientation vector is unknown. Results are also stored in the '`pol_vector`' property of the array object.	
Input	element	The element numbers for which this functions is applied. If no element number is given, the common phase is estimated for all elements in the array.
	verbose	Enables (1, default) or disables (0) the progress bar.
Output	pol_vector	The orientation of the electric field in 3D local cartesian coordinates.
	err	provides information on how well the pol-vector matches the pattern. It scales from 0 to 1 where 0 is a perfect match.

generate (array_type, element, phi_3dB, theta_3dB, rear_gain)		
Description	Generates predefined arrays.	
Array Types	omni	An isotropic radiator with vertical polarization.
	dipole	A short dipole radiating with vertical polarization.
	half-wave-dipole	A half-wave dipole radiating with vertical polarization.
	patch	A vertically polarized ideal patch antenna with 90° opening in azimuth and elevation.
	custom	An antenna with a custom gain in elevation and azimuth. E.g.: '`a.generate('custom',1,90,90,0.1)`' creates an array with 90° opening in azimuth and elevation and 0.1 rear gain.
	xpol	Two elements with ideal isotropic patterns (vertical polarization). The second element is tilted by 90°.
	rhcp-dipole	Two crossed dipoles with one port. The signal on the second element (horizontal) is shifted by -90° out of phase. The two elements thus create a right hand circular polarized (RHCP) signal.
	lhcp-dipole	Two crossed dipoles with one port. The signal on the second element (horizontal) is shifted by 90° out of phase. The two elements thus create a left hand circular polarized (LHCP) signal.
	lhcp-rhcp-dipole	Two crossed dipoles. For input port 1, the signal on the second element is shifted by +90° out of phase. For input port 2, the the signal on the second element is shifted by -90° out of phase. Port 1 thus transmits a LHCP signal and port 2 transmits a RHCP signal.
	ula2	Unified linear arrays composed of 2 omni-antennas (vertical polarization) with 10 cm element distance.
	ula4	Unified linear arrays composed of 4 omni-antennas (vertical polarization) with 10 cm element distance.
	ula8	Unified linear arrays composed of 8 omni-antennas (vertical polarization) with 10 cm element distance.
Input	array_type	One of the above array types.
	element	The element numbers for which this functions is applied. If no element number is given, the function creates a new array and delete the old elements in the array.
	phi_3dB	The 3dB beam width in azimuth direction (used only for '`custom`' array type)
	theta_3dB	The 3dB beam width in elevation direction (used only for '`custom`' array type)
	rear_gain	The isotropic gain (linear scale) at the back of the antenna (used only for '`custom`' array type)

eMail: quadriga@hhi.fraunhofer.de

[h_array, mse, mse_pat] = **array.import_pattern** (fVi, fHi , correct_phase, accuracy, max_num_elements, azimuth_grid, elevation_grid, verbose)

Description	This function converts any antenna field pattern into a QuaDRiGa antenna array object. The conversion needs at most 4 elements for each input field pattern. This value is doubled if the inputs are complex-valued.	
Input	fVi	The field pattern(s) for the vertical polarization given in spherical coordinates. The first dimension corresponds to the elevation angle (ranging from -90 to 90 degrees). The second dimension is for the azimuth angle (ranging from -180 to 180 degrees). The third dimension belongs to the element number. The default resolution is 1 degree. Hence, the default size of fVi is ¡181x361x1¿. If a different resolution is given, the optional variables '`azimuth_grid`' and '`elevation_grid`' must be defined.
	fHi	The field pattern(s) for the horizontal polarization given in spherical coordinates. '`fHi`' can be empty if no horizontal response is given. If it is given, then '`fHi`' must have the same size as '`fVi`'.
	correct_phase	It is possible to remove a common phase offset for both, vertical and horizontal polarization. This offset can occur when an antenna was characterized in an anechoic chamber where the antenna was not placed exactly in the center. Hence, there is a phase difference for each angle. Default setting: 0 (No correction)

	accuracy	The required accuracy (i.e. the target mean square error (MSE)) of the approximation. The MSE is defined as:
		`N = abs( fVi ).^2 + abs( fHi ).^2;`
		`offset = abs( fVi - fVo ).^2 + abs( fHi - fHo ).^2;`
		`MSE = mean( offset(:) ) / N;`
		This variable is tracked throughout the converting process. If the approximation is good enough (i.e. the MSE < accuracy), then the algorithm stops and returns the output pattern. Default value: 1e-6

	max_num_elements	This value limits the number of elements per field pattern. The default value is 4 which allows a perfect approximation. However, more elements require more computing time in the channel model.
	azimuth_grid	Azimuth angles in [rad] were samples of the field patterns are provided The field patterns are given in spherical coordinates. This variable provides the azimuth sampling angles in radians ranging from $-\pi$ to π.
	elevation_grid	Elevation angles in [rad] were samples of the field patterns are provided The field patterns are given in spherical coordinates. This variable provides the elevation sampling angles in radians ranging from $-\frac{\pi}{2}$ (downwards) to $\frac{\pi}{2}$ (upwards).
	verbose	Enables (1, default) or disables (0) the progress bar.
Output	h_array	The QuaDRiGa antenna array object generated from the field patterns.
	mse	The MSE (as defined above) for each pattern.
	mse_pat	The MSE (as defined above) for each pattern given for each sample point of the pattern in spherical coordinates.

[V, H, CP, dist] = **interpolate** (azimuth, elevation, element)		
Description	Interpolates the field pattern Note: Interpolation of the beam patterns is very computing intensive. It must be performed several thousands of times during a simulation run. It is highly recommended to use linear interpolation for this task since this method is optimized for speed. Spline interpolation calls the MATLAB internal interpolation function which is more than 10 times slower. To enable linear interpolation, set the '`interpolation_method`' property of the array object to '`linear`'. Remark: There are additional input parameters specified in the .mat-File that are not in the list below. Those parameters correspond to the properties of the '`array`' class. Passing those variables during the function call takes less time than reading them from the object properties. This is used internally in '`channel_builder.get_channels`' but is irrelevant here.	
Input	azimuth elevation element	A vector of azimuth angles in [rad]. A vector of elevation angles in [rad]. The element numbers for which this interpolation is done is applied. If no element number is given, the interpolation is done for all elements in the array.
Output	V H CP dist	The interpolated vertical field pattern. The interpolated horizontal field pattern. The interpolated common phase field pattern. The effective distances between the antenna elements when seen from the direction of the incident path. The distance is calculated by an projection of the array positions on the normale plane of the incident path.

cp = **rotate_pattern** (deg, rotaxis, element, usage)		
Description	Rotates antenna patterns Pattern rotation provides the option to assemble antenna arrays out of single elements. By setting the '`element_position`' property of an array object, elements can be placed at different coordinates. In order to freely design arbitrary array configurations, however, elements often need to be rotated (e.g. to assemble a +/- 45° crosspolarized array out of single dipoles). This functionality is provided here. Note: Calling '`rotate_pattern`' will always remove the common phase from the field patterns. Call '`estimate_common_phase`' before calling '`rotate_pattern`' to extract the common phase information.	
Input	deg rotaxis element usage	The rotation angle in [degrees] ranging from -180° to 180°. The rotation axis specified by the character 'x','y', or 'z'. The element numbers for which this interpolation is done is applied. If no element number is given, the interpolation is done for all elements in the array. The optional parameter 'usage' can limit the rotation procedure either to the pattern or polarization. Possible values are: • 0: Rotate both (pattern+polarization) - default • 1: Rotate only pattern • 2: Rotate only polarization
Output	cp	The common phase of the field pattern.

set_grid (azimuth_grid, elevation_grid)		
Description	Sets a new grid for azimuth and elevation and interpolates the pattern This function replaces the properties '`azimuth_grid`' and '`elevation_grid`' of the antenna object with the given values and interpolates the antenna patterns to the new grid.	
Input	azimuth_grid elevation_grid	Azimuth angles in [rad] were samples of the field patterns are provided The field patterns are given in spherical coordinates. This variable provides the azimuth sampling angles in radians ranging from $-\pi$ to π. Elevation angles in [rad] were samples of the field patterns are provided The field patterns are given in spherical coordinates. This variable provides the elevation sampling angles in radians ranging from $-\frac{\pi}{2}$ (downwards) to $\frac{\pi}{2}$ (upwards).

visualize (element)		
Description	Create a plot showing the element configurations.	
Input	element	The element numbers for which the plot os created. If no element number(s) are given, a plot is created for each element in the array.

2.2.3 Class "track"

One feature of the channel model is the continuous evolution of wireless channels when the terminal moves through the environment. A track describes the movement of a mobile terminal. It is composed of an ordered list of positions. During the simulation, one snapshot is generated for each position on the track.

Along the track, wireless reception conditions may change, e.g. when moving from an area with LOS to a shaded area. This behavior is described by segments, or states. A segment is a subset of positions that have similar reception conditions. Each segment is classified by a segment index (i.e. the center position of the segment) and a scenario. The scenario must be one of the supported scenarios in class '`parameter_set`'.

Properties

name	Name of the track
initial_position	Position offset (will be added to positions) This position is given in global cartesian coordinates (x,y, and z-component) in units of [m]. The initial position normally refers to the starting point of the track. If the track has only one segment, it is also the position for which the LSPs are calculated. The initial position is added to the values in the positions variable.
no_snapshots	Number of positions on the track
positions	Ordered list of position relative to the initial position QuaDRiGa calculates an instantaneous channel impulse response (also called snapshot) for each position on the track.

movement_profile	Time (in sec) vs. distance (in m) for speed profile
	QuaDRiGa supports variable terminal speeds. This is realized by interpolating the channel coefficients at the output of the model. The variable '`track.movement_profile`' describes the movement along the track by associating a time-point with a distance-point on the track. An example is:
	`t.movement_profile = [ 0,7 ; 5,0 ; 6,0 ; 20,20 ]';`
	`dist = t.interpolate_movement( 1e-3 );`
	`ci = cn.interpolate( dist , t.get_length );`
	See also the tutorial "Applying Varying Speeds (Channel Interpolation)" in Section A.6 for more details.

no_segments	Number of segments or states along the track
segment_index	Starting point of each segment given as index in the '`positions`' vector
scenario	Scenarios for each segment along the track This variable contains the scenario names for each segment as a cell array of strings. A list of supported scenarios can be obtained by calling '`parameter_set.supported_scenarios`'. If there is only one transmitter (i.e. one base station), the cell array has the dimension [1 x no_segments]. For multiple transmitters, the rows of the array may contain different scenarios for each transmitter. For example, in a multicell setup with three terrestrial base stations, the propagation conditions may be different to all BSs. The cell arrays than has the dimension [3 x no_segments].
par	Manual parameter settings This variable contains a structure with LSPs. This can be used for assigning LSPs directly to the channel builder, e.g. when they are obtained from measurements. The structure contains the following fields: • ds - The delay spread in [s] per segment • kf - The Ricean K-Factor in [dB] per snapshot • pg - The effective path gain in [dB] excluding antenna gains per snapshot • asD - The azimuth angle spread in [deg] per segment at the transmitter • asA - The azimuth angle spread in [deg] per segment at the receiver • esD - The elevation angle spread in [deg] per segment at the transmitter • esA - The elevation angle spread in [deg] per segment at the receiver • xpr - The NLOS cross-polarization in [dB] per segment If there is only a subset of variables (e.g. the angle spreads are missing), then the corresponding fields can be left empty. They will be completed by the parameter sets. See also the method '`track.generate_parameters`' on how to fill this structure automatically.
ground_direction	Azimuth orientation of the terminal antenna for each snapshot This variable can be calculated automatically from the positions by the function '`track.compute_directions`'.

height_direction	Elevation orientation of the terminal antenna for each snapshot
closed	Indicates that the track is a closed curve

Methods

h_track = **track** (track_type, track_length, direction, street_length_min, street_length_mu, street_length_std, curve_radius, turn_probability)	
Description	Creates a new track object. See '`track.generate`' for a description of the input parameters and the list of supported track types.

compute_directions	
Description	Calculates ground and height orientations from positions This function calculates the orientations of the transmitter based on the positions. If we assume that the receive antenna array is fixed on a car and the car moves along the track, then the antenna turns with the car when the car is changing direction. This needs to be accounted for when generating the channel coefficients. This function calculates the orientation based on the positions and stored the output in the ground_direction and height_direction field of the track object.

copy_objects	
Description	A modified version of the standard physical copy function While the standard copy command creates new physical objects for each element of obj (in case obj is an array of object handles), copy_objects checks whether there are object handles pointing to the same object and keeps this information.

correct_overlap (overlap)		
Description	Corrects positions of the segment start to account for the overlap. After the channel coefficients are calculated, adjacent segments can be merged into a time-continuous output. The merger assumes that the merging interval happens at the end of one segment, before a new segments starts. In a reality, however, the scenario change happens in the middle of the overlapping part (and not at the end of it). This function corrects the position of the segment start to account for that.	
Input	overlap	The length of the overlapping part relative to the segment length. It can have values in between 0 (no overlap) and 1 (ramp along the entire segment).

generate ('linear', track_length, direction)		
Description	Creates a linear track with given length and direction. Direction describes the travel direction along the track in [rad] in mathematical sense (i.e. 0 means east, pi/2 means north, pi means west and -pi/2 south). If "track_length" or "direction" is not specified, then the default track is 1 m long and has a random direction.	
Input	track_length direction	The track length in [m]. Default length is 1 m. specifies the driving direction in [rad]. The default is random.

generate ('circular', track_length, direction)		
Description	Creates a circular track with given length and starting-direction.	
Input	track_length direction	The circumference of the circular track in [m]. Default is 62.8 m. The starting point on the circle in [rad]. Positive values define the travel direction as counter clock-wise and negative values as clock-wise. E.g. 0 sets the start point in the east of the circle, traveling north; -2π sets it in the east, traveling south. The default is random.

generate ('street', track_length, direction, street_length_min, street_length_mu, street_length_std, curve_radius, turn_probability)		
Description	Emulates a drive route through a city grid. The mobile terminal starts at point 0, going into a specified direction. The trajectory grid is build from street segments. The length of each street is specified by the parameters '`street_length_min`', '`street_length_mu`', and '`street_length_sigma`'. At the end of a street (i.e. at a crossing), the terminal turns with a probability specified by '`turn_probability`'. The change of direction is in between 75 and 105 degrees either left or right. The radius if the curve is given by '`curve_radius`'. The track is set up in a way that prevents driving in circles.	
Input	track_length	the length in [m]. Default length is 1000 m.
	direction	specifies the driving direction in [rad] of the first segment in mathematical sense (0 means east, pi/2 means north). The default value is random.
	street_length_min	the minimal street length in [m]. The default is 50 m.
	street_length_mu	the median street length in [m]. The default is 187 m. This value was obtained from measurements in Berlin, Germany.
	street_length_std	the standard deviation of the street length in [m]. The default is 83 m. This value was obtained from measurements in Berlin, Germany.
	curve_radius	the curve radius during a turn in [m]. The default is 10 m.
	turn_probability	the probability of a turn at a crossing. Possible values are in between 0 and 1. The default is 0.5.

par = **generate_parameters** (overlap, usage, check_parfiles, verbose)		
Description	Generates large scale parameters and stores them in '`par`' This function extracts the LSPs for the given scenario from the '`parameter_set`' class and stores them in '`track.par`'. Hence, it automatically generates the LSPs and, thus, implements an easy-to-use interface for the '`parameter_set`' class. Since the track class does not handle transmitter positions, a default position of [0,0,25] is assumed. Please refer to '`layout.generate_parameters`' for a more detailed description.	
Input	overlap	The length of the overlapping part relative to the segment length. When there are scenario transitions, KF and PG change smoothly during a predefined interval. The length of that interval is a percentage of previous segment. The parameter overlap adjusts this percentage, ranging from 0 (i.e. very hard, step-like change at the scenario boundary) to 1 (very smooth, but long transition).
	usage	Changes the behavior of the method usage = 0 Deletes all existing parameters from the track. usage = 1 Deletes all existing parameters from the track and generates new ones. Existing LSPs will be overwritten. usage = 2 (default) Keeps existing parameters, but generates missing ones. This is useful when, for example, the effective path gain (PG) is provided, but the other LSPs are unknown. In this case, the unknown "gaps" are filled with values which are generated from the provided scenario description.
	check_parfiles	check_parfiles = 0 / 1 (default: 1) Disables (0) or enables (1) the parsing of shortnames and the validity-check for the config-files. This is useful, if you know that the parameters in the files are valid. In this case, this saves execution time.
	verbose	Enables (1, default) or disables (0) the progress bar.
Output	par	The automatically generated parameters This variable contains structure of the LSPs with the following fields: • ds - The delay spread in [s] per segment • kf - The Ricean K-Factor in [dB] per snapshot • pg - The effective path gain in [dB] excluding antenna gains per snapshot • asD - The azimuth angle spread in [deg] per segment at the transmitter • asA - The azimuth angle spread in [deg] per segment at the receiver • esD - The elevation angle spread in [deg] per segment at the transmitter • esA - The elevation angle spread in [deg] per segment at the receiver • xpr - The NLOS cross-polarization in [dB] per segment An identical copy of this variable is assigned to '`track.par`'.

[len, dist] = **get_length**		
Description	Calculates the length of the track in [m]	
Output	len dist	Length of a track in [m] Distance of each position (snapshot) from the start of the track in [m]

subtracks = **get_subtrack** (i_segment)		
Description	Splits the track in subtracks for each segment (state) This function returns the subtracks for the given segment indices. When no input argument is provided, all subtracks are returned. After defining segments along the track, one needs the subtrack that corresponds only to one segment to perform the channel calculation. This new track can consist of two segments. The first segment contains the positions from the previous segment, the second from the current. This is needed to generate overlapping channel segments for the merging process.	
Input	i_segment	A list of indices indicating which subtracks should be returned. By default, all subtracks are returned.
Output	subtracks	A vector of track objects corresponding to the number of segments.

dist = **interpolate_movement** (si, method)		
Description	Interpolates the movement profile to a distance vector This function interpolates the movement profile. The distance vector at the output can then be used to interpolate the channel coefficients to emulate varying speeds. See also the tutorial "Applying Varying Speeds (Channel Interpolation)" in Section A.6 for more details.	
Input	si method	the sampling interval in [seconds]. selects the interpolation algorithm. The default is cubic spline interpolation. Optional are: • nearest - Nearest neighbor interpolation • linear - Linear interpolation • spline - Cubic spline interpolation • pchip - Piecewise Cubic Hermite Interpolating Polynomial • cubic - Cubic spline interpolation
Output	dist	Distance of each interpolated position from the start of the track in [m]

interpolate_positions (samples_per_meter)		
Description	Interpolates positions along the track This function interpolates the positions along the track such that it matches the samples per meter specifies in the simulation parameters. The channel model operates on a position-based sample grid. That means that the '`channel_builder`' generates one CIR for each position on the track. In practise, however, a time continuous evolution of the CIR is often needed. This can be interpolated from the position-based grid, provided that the spatial sample theorem is not violated (i.e. the channel needs to be sampled at least twice per half wave length). In order to do that, enough sample points are needed along the track. INTERPOLATE_POSITIONS calculates the missing sample points and places them equally spaced along the track. This corresponds to a constant speed when evaluating the output CIRs. The required value for '`samples_per_meter`' can be obtained from the '`simulation_parameters`' object.	
Input	samples_per_meter	the samples per meter (e.g. from '`simulation_parameters.samples_per_meter`').

set_scenario (scenario, probability, seg_length_min, seg_length_mu, seg_length_std)		
Description	Assigns random scenarios and creates segments. This function can be used to create segments along the trajectory and assign scenarios to the segments. If there are less than 3 input arguments (i.e. only '`scenario`' and/or '`probability`' is given), then no segments will be created. To create segments with the default settings call '`set_scenario(scenario,[],[])`'.	
Input	scenario	A cell array of scenario-names. Each scenario (synonym for propagation environment) is described by a string (e.g. "MIMOSA_16-25_LOS" or "WINNER_SMa_C1_NLOS"). A list of supported scenarios can be obtained by calling '`parameter_set.supported_scenarios`'. The scenario parameters are stored in the configuration folder "config" in the QuaDRiGa main folder. The filenames (e.g. "MIMOSA_16-25_LOS.conf") also serves as scenario name.
	probability	The probability for which the scenario occurs. This parameter must be a vector of the same length as there are scenarios. Probabilities must be specified in between 0 and 1. The sum of the probabilities must be 1. By default (or when '`probability`' is set to '`[]`'), each scenario is equally likely.
	seg_length_min	the minimal segment length in [m]. The default is 10 m.
	seg_length_mu	the median segment length in [m]. The default is 30 m.
	seg_length_std	the standard deviation of the street length in [m]. The default is 12 m.

set_speed (speed)		
Description	Sets a constant speed in [m/s] for the entire track. This function fills the '`track.movement_profile`' field with a constant speed value. This helps to reduce computational overhead since it is possible to reduce the computation time by interpolating the channel coefficients.	
Input	speed	The terminal speed in [m/s]

split_segment (mi, ma, mu, sig, no_check)		
Description	Splits long segments in subsegments of the same type.	
Input	mi	Minimum length of the subsegment in [m], default: 10m
	ma	Maximum length of the subsegment in [m], must be > 2*mi, default: 30m
	mu	Mean length of the subsegment (mi < mu < ma), default: 15m
	sig	Std of the length of the subsegment, default: 5m
	no_check	Disable parsing of input variables, default: false

visualize	
Description	Plots the track.

2.2.4 Class "layout"

Objects of this class define the network layout of a simulation run. Each network layout has one or more transmitters and one or more receivers. Each transmitter and each receiver need to be equipped with an antenna array which is defined by the array class. In general, we assume that the transmitter is at a fixed position and the receiver is mobile. Thus, each receivers movement is described by a track.

Properties

name	Name of the layout
simpar	Handle of a 'simulation_parameters' object. See Section 2.2.1
no_tx	Number of transmitters (or base stations)
no_rx	Number of receivers (or mobile terminals)
tx_name	Identifier of each Tx, must be unique
tx_position	Position of each Tx in global cartesian coordinates using units of [m]
tx_array	Handles of 'array' objects for each Tx. See Section 2.2.2
rx_name	Identifier of each Tx, must be unique
rx_position	Initial position of each Rx (relative to track start) in global cartesian coordinates using units of [m]
rx_array	Handles of 'array' objects for each Rx. See Section 2.2.2
track	Handles of track objects for each Rx. See Section 2.2.3
pairing	An index-list of links for which channel are created. The first row corresponds to the Tx and the second row to the Rx.
no_links	Number of links for which channel coefficients are created (read only)

Methods

h_layout = **layout** (simpar)		
Description	Creates a new layout object.	
Input	simpar	Handle of a '`simulation_parameters`' object. See Section 2.2.1

[h_parset, h_cb] = **create_parameter_sets** (initialize, check_parfiles)		
Description	Creates '`parameter_set`' objects based on layout specification This function processes the data in the '`layout`' object. First, all tracks in the layout are split into subtracks. Each subtrack corresponds to one segment. Then, then scenario names are parsed. A '`parameter_set`' object is created for each scenario and for each transmitter. For example, if there are two terrestrial BSs, each having urban LOS and NLOS users, then 4 '`parameter_set`' objects will be created (BS1-LOS, BS2-NLOS, BS2-LOS, and BS2-NLOS). The segments are then assigned to the '`parameter_set`' objects. In the last step, the parameter maps are created (see Section 3.1.4). This can be disabled by setting '`initialize = 0`'.	
Input	initialize	Enables (1, default) or disables (0) the generation of the parameter maps. If you want to adjust the parameters first, use '`initialize = 0`', then adjust the parameters in the '`parameter_set`' objects and call '`update_parameters`' manually.
	check_parfiles	Enables (1, default) or disables (0) the parsing of shortnames and the validity-check for the config-files. This is useful, if you know that the parameters in the files are valid. In this case, this saves some execution time.
Output	h_parset	A matrix of '`parameter_set`' objects. Rows correspond to the scenarios, columns correspond to the transmitters. See Section 2.2.5
	h_cb	A vector of '`channel_builder`' objects. See Section 2.2.6

h_layout = **layout.generate** ('regular', no_sites, isd, h_array)		
Description	generates a new multicell layout using a regular grid of BS positions. Each BS has three sectors.	
Input	no_sites	the number of sites. This can be be 1, 7 or 19 - resulting in 3, 21 or 57 sectors, respectively.
	isd	the inter-site distance between the BSs in [m].
	h_array	the antenna array. '`h_array`' is for one sector only. It will be rotated to match the sector orientations and copied to all sites. The broadside-direction of the provided antenna must be 0 (facing east).
Output	h_layout	The generated layout.

[par, h_parset] = **generate_parameters** (overlap, usage, check_parfiles)		
Description	Generates large scale parameters and stores them in '`track.par`' Normally, parameters are handled by objects of the '`parameter_set`' class, which are generated by calling '`layout.create_parameter_sets`'. Those objects then feed the parameters to the '`channel_builder`'. However, this method is rather inflexible when the user wants to manipulate the parameters directly. As an alternative, parameters can be provided in the property '`track.par`' of the '`track`' class. This allows the user to edit parameters without dealing with the '`parameter_set`' objects. This function extracts the LSPs for the given scenario from the '`parameter_set`' class and stores them in '`track.par`'. Hence, it automatically generates the LSPs and, thus, implements an easy-to-use interface for the '`parameter_set`' class.	
Input	overlap	The length of the overlapping part relative to the segment length. When there are scenario transitions, KF and PG change smoothly during a predefined interval. The length of that interval is a percentage of previous segment. The parameter overlap adjusts this percentage, ranging from 0 (i.e. very hard, step-like change at the scenario boundary) to 1 (very smooth, but long transition).
	usage	Changes the behavior of the method usage = 0 Deletes all existing parameters from the track. usage = 1 Deletes all existing parameters from the track and generates new ones. Existing LSPs will be overwritten. usage = 2 (default) Keeps existing parameters, but generates missing ones. This is useful when, for example, the effective PG is provided, but the other LSPs are unknown. In this case, the unknown "gaps" are filled with values which are generated from the provided scenario description.
	check_parfiles	check_parfiles = 0 / 1 (default: 1) Disables (0) or enables (1) the parsing of shortnames and the validity-check for the config-files. This is useful, if you know that the parameters in the files are valid. In this case, this saves execution time.
Output	par	The automatically generated parameters. This cell array contains a parameter structure of the LSPs for each receiver with the following fields: • ds - The delay spread in [s] per segment • kf - The Ricean K-Factor in [dB] per snapshot • pg - The effective path gain in [dB] excluding antenna gains per snapshot • asD - The azimuth angle spread in [deg] per segment at the transmitter • asA - The azimuth angle spread in [deg] per segment at the receiver • esD - The elevation angle spread in [deg] per segment at the transmitter • esA - The elevation angle spread in [deg] per segment at the receiver • xpr - The NLOS cross-polarization in [dB] per segment An identical copy of this variable is assigned to '`track.par`'.
	h_parset	A matrix of '`parameter_set`' objects. Rows correspond to the scenarios, columns correspond to the transmitters. See Section 2.2.5

[h_channel, h_parset, h_cb] = **get_channels** (sampling_rate, check_parfiles)		
Description	Calculate the channel coefficients. This is the most simple way to create the channel coefficients. This function executes all steps that are needed to generate the channel coefficients. Hence, it is not necessary to use use the '`parameter_set`' or '`channel_builder`' objects.	
Input	sampling_rate	channel update rate in [s]. This parameter is only used if a speed profile is provided in the track objects. Default value: 0.01 = 10 ms
	check_parfiles	Enables (1, default) or disables (0) the parsing of shortnames and the validity-check for the config-files. This is useful, if you know that the parameters in the files are valid. In this case, this saves some execution time.
Output	h_channel	A vector channel objects. See Section 2.2.7
	h_parset	A matrix of '`parameter_set`' objects. Rows correspond to the scenarios, columns correspond to the transmitters. See Section 2.2.5
	h_cb	A vector of '`channel_builder`' objects. See Section 2.2.6

[map, posx, posy] = **power_map** (scenario, usage, res, xs, ys, xe, ye, tx_power)		
Description	Calculates a power-map for the given layout. This functions calculates the power seen by a terminal for a given layout. This helps to predict the performance and for a given setup.	
Input	scenario	The scenario for which the map shall be created. There are four options: 1. A string describing the scenario. A list of supported scenarios can be obtained by calling '`parameter_set.supported_scenarios`'. 2. cell array of strings describing the scenario for each transmitter in the layout. 3. A '`parameter_set`' object. This method is useful if you need to edit the parameters first. For example: call '`p = parameter_set(UMal)`' to load the parameters. Then edit '`p.scenpar`' or '`p.plpar`' to adjust the settings. 4. Aa array of '`parameter_set`' objects describing the scenario for each transmitter in the layout.
	usage	A string specifying the detail level. The following options are implemented: • '`quick`' - Uses the antenna patterns, the LOS path, and the path gain from the scenario • '`sf`' - Uses the antenna patterns, the LOS path, the path gain from the scenario, and a shadow fading map • '`detailed`' - Runs a full simulation for each pixel of the map (very slow) • '`phase`' - Same as quick, but the output contains the complex-valued amplitude instead of the power
	res	Distance between sample points in [m] (default = 10 m)
	xs	x-coordinate in [m] of the top left corner
	ys	y-coordinate in [m] of the top left corner
	xe	x-coordinate in [m] of the bottom right corner
	ye	y-coordinate in [m] of the bottom right corner
	tx_power	A vector of tx-powers in [dBm] for each transmitter in the layout. This power is applied to each transmit antenna in the tx-antenna array. By default (if `tx_power` is not given), 0 dBm are assumed.
Output	map	A cell array containing the power map for each transmitter in the layout. The maps have the dimensions [y-coordinate , x-coordinate , Rx-antenna , Tx-antenna].
	posx	Vector with the x-coordinates of the map.
	posy	Vector with the y-coordinates of the map.

randomize_rx_positions (max_dist, min_height, max_height, track_length)		
Description	Generates random Rx positions and tracks. Places the users in the layout at random positions. Each user will be assigned a linear track with random direction. The random height of the user terminal will be in between '`min_height`' and '`max_height`'.	
Input	max_dist	the maximum distance from the layout center in [m]. Default is 50 m.
	min_height	the minimum user height in [m]. Default is 1.5 m.
	max_height	the maximum user height in [m]. Default is 1.5 m.
	track_length	the length of the linear track in [m]. Default is 1 m.

[pairs, power] = **set_pairing** (method, threshold, tx_power, check_parfiles)		
Description	Determines links for which channel coefficient are generated. This function can be used to automatically determine the links for which channel coefficients should be generated. For example, in a large network there are multiple base stations and mobile terminals. The base stations, however, only serve a small area. It the terminal is far away from this area, it will receive only noise from this particular BS. In this case, the channel coefficients will have very little power and do not need to be calculated. Disabling those links can reduce the computation time and the storage requirements for the channel coefficients significantly. There are several methods to du this which can be selected by the input variable '`method`'.	
Methods	'all'	Enables the simulation of all links.
	'power'	Calculates the expected received power taking into account the path loss, the antenna patterns, the LOS polarization, and the receiver orientation. If the power of a link is below the '`threshold`', it gets deactivated.
	'sf'	Same as '`power`', but this option also includes the shadow fading. Therefore, the LSPs have to be calculated. LSPs get then stored in '`layout.track.par`'. This method is the most accurate. The actual power in the channel coefficients can be up to 6 dB higher due to multipath effects.
Input	method	Link selection method. Supported are: 'all', 'power', and 'sf' (see above).
	threshold	If the Rx-power is below the threshold in [dBm], the link gets deactivated.
	tx_power	A vector of tx-powers in [dBm] for each transmitter in the layout. This power is applied to each transmit antenna in the tx-antenna array. By default (if '`tx_power`' is not given), 0 dBm are assumed.
	check_parfiles	Disables (0) or enables (1, default) the parsing of shortnames and the validity-check for the config-files. This is useful, if you know that the parameters in the files are valid. In this case, this saves execution time.
Output	pairs	An index-list of links for which channel are created. The first row corresponds to the Tx and the second row to the Rx. An identical copy gets assigned to '`layout.pairing`'.
	power	A matrix containing the estimated receive powers for each link in [dBm]. Rows correspond to the receiving terminal, columns correspond to the transmitter station. For MIMO links, the power of the strongest MIMO sublink is reported.

pos = **set_satellite_pos** (rx_latitude, sat_el, sat_az, sat_height, tx_no)		
Description	Calculates the Tx position from a satellite orbit. QuaDRiGas reference coordinate system is on the surface of the earth. In order to use QuaDRiGa for satellite links, the satellite position must be set. Normally, this position is given in azimuth and elevation relative to the users position. This function takes a satellite orbital position and calculates the corresponding transmitter coordinates.	
Input	rx_latitude	The receiver latitude coordinate on the earth surface in [deg]. Default is 52.5.
	sat_el	Satellite elevation in [deg]. Default is 31.6.
	sat_az	Satellite azimuth in [deg] given in compass coordinates. Default is 180° (south).
	sat_height	Satellite height in [km] relative to earth surface. Default is 35786 (GEO orbit).
	tx_no	The 'tx_no' in the layout object for which the position should be set. Default is 1.
Output	pos	The satellite positions in the metric QuaDRiGa coordinate system.

visualize	
Description	Plots the layout.

2.2.5 Class "parameter_set"

This class implements all functions that are necessary to generate and manage correlated LSPs. It also provides interfaces for the channel builder. LSPs are the shadow fading, the Ricean K-Factor, the RMS delay spread and the four angles (elevation and azimuth at the transmitter and receiver). This class implements some core functions of the channel model and the user does normally not need to interact with it. However, if parameter tables need to be changed, here is the place to do so.

Properties

name	Name of the '`parameter_set`' object
simpar	Handle of a '`simulation_parameters`' object. See Section 2.2.1
tx_array	Handles of '`array`' objects for each Tx. See Section 2.2.2
rx_array	Handles of '`array`' objects for each Rx. See Section 2.2.2
rx_track	Handles of '`track`' objects for each Rx. See Section 2.2.3
scenario	Name of the scenario (text string)
scenpar	The parameter table. See Section 2.4
plpar	Parameters for the path loss. See Section 2.4
no_positions	Number of receiver positions associated to this '`parameter_set`' object Note that each segment in longer tracks is considered a new Rx position.
positions	The list of initial positions for which LSPs are generated This variable is obtained from the properties '`track.initial_position`' and '`layout.rx_position`'
tx_position	The transmitter position obtained from the corresponding '`layout.tx_position`'
ds	The RMS delay spread in [s] for each receiver position
kf	The Ricean K-Factor [linear scale] for each receiver position
sf	The shadow fading [linear scale] for each receiver position
asD	The azimuth spread of departure in [deg] for each receiver position
asA	The azimuth spread of arrival in [deg] for each receiver position
esD	The elevation spread of departure in [deg] for each receiver position
esA	The elevation spread of arrival in [deg] for each receiver position
xpr	The cross polarization ratio [linear scale] for each receiver position
parameter_maps	The seven large-scale parameter maps in logarithmic scale Rows correspond to the y-coordinate and columns to the x-coordinate. The order of the maps (third dimension) is: DS, KF, SF, asD, asA, esD, esA. All values are logarithmic.
map_extension	Distance in [m] that is added to each direction when generating maps
map_extent	Extent of the mpas in x- and y-direction [xmin, xmax; ymin, ymax] in [m]
map_size	Number of map pixels in x and y-direction [n_x_samples; n_y_samples]
samples_per_meter	Resolution of the decorrelation maps in [samples/m] This value is obtained from '`simulation_parameters.map_resolution`'
map_valid	Indicates if maps contain valid data
LSP_xcorr_matrix	The Cross-correlation matrix for the LSPs
LSP_matrix_isOK	Determines if the XCorr-matrix is positive definite
map_x_coord	The x-coordinates in [m] for each pixel of the maps
map_y_coord	The y-coordinates in [m] for each pixel of the maps

Methods

h_parset = **parameter_set** (scenario, positions, check_parfiles)

Description	Creates a new '`parameter_set`' object.	
Input	scenario	The scenario name for which the parameters should be loaded. A list of supported scenarios can be obtained by calling '`parameter_set.supported_scenarios`'.
	positions	The list of initial positions for which LSPs should be generated.
	check_parfiles	check_parfiles = 0 / 1 (default: 1) Disables (0) or enables (1) the parsing of shortnames and the validity-check for the config-files. This is useful, if you know that the parameters in the files are valid. In this case, this saves execution time.
Output	h_parset	A '`parameter_set`' object.

copy_objects		
Description	A modified version of the standard physical copy function While the standard copy command creates new physical objects for each element of obj (in case obj is an array of object handles), copy_objects checks whether there are object handles pointing to the same object and keeps this information.	

angles = **get_angles**		
Description	Calculates the departure- and arrival angles of the LOS path between Tx and Rx.	
Output	angles	A matrix containing the four angles: • Azimuth of Departure at the Tx (AoD, row 1) • Azimuth of Arrival at the Rx (AoA, row 2) • Elevation of Departure at the Tx (EoD, row 3) • Elevation of Arrival at the Rx (EoA, row 4) The number of columns corresponds to the number of rx-positions.

[h_channel, h_cb] = **get_channels**		
Description	Calculate the channel coefficients.	
Output	h_channel h_cb	A vector channel objects. See Section 2.2.7 A vector of '`channel_builder`' objects. See Section 2.2.6

dist = **get_distances**		
Description	Calculates the distances between Rx and Tx.	
Output	dist	A vector containing the distances between each Rx and the Tx in [m].

[pl, scale_sf] = **get_pl** (evaltrack, i_mobile)		
Description	Calculates the path loss. The path loss model is specified in the configuration files and in '`parameter_set.plpar`'.	
Input	evaltrack	A '`track`' object for which the PL should be calculated. If '`evaltrack`' is not given, then the path loss is calculated for each Rx position. Otherwise the path loss is calculated for the positions provided in '`evaltrack`'.
	i_mobile	The Rx-index. If it is not given, the PL is evaluated for all Rx positions. If '`evaltrack`' is given, and if '`simulation_parameters.drifting_precision`' is set to 3, then this parameter is required to select the Rx antenna array (default = 1).
Output	pl scale_sf	The path loss in [dB] In some scenarios, the SF might change with increasing distance between Tx and Rx. Hence, the shadow fading provided by the parameter map has to be changed accordingly. The second output parameter "`scale_sf`" can be used for scaling the (logarithmic) SF value from the map.

[sf,kf] = **get_sf_profile** (evaltrack, i_mobile)		
Description	This function returns the shadow fading and the K-factor along the given track. This function is mainly used by the channel builder class to scale the output channel coefficients. The profile is calculated by using the data in the correlation maps and interpolating it to the positions in the given track. Increasing the resolution of the maps also increases the resolution of the profile.	
Input	evaltrack	A '`track`' object for which the SF and KF should be interpolated.
	i_mobile	If '`simulation_parameters.drifting_precision`' is set to 3, then this parameter is required to select the Rx antenna array.
Output	sf kf	The shadow fading [linear scale] along the track. The K-factor [linear scale] along the track.

set_par (name, value)		
Description	Sets the parameters of all objects in '`parameter_set`' arrays. This function sets all values of the parameter specified by '`name`' of the '`parameter_set`'-array to the given value. Example: '`set_par`('ds' , 1e-9) sets all ds-values to 1 ns.	
Input	name	The fieldname that should be altered. Supported are: 'ds', 'kf', 'sf', 'asD', 'asA', 'esD', 'esA', 'samples_per_meter', 'map_extension', and 'LSP_xcorr_matrix'.
	value	The value that should be assigned. If the '`LSP_xcorr_matrix`' is altered, then the lower triangular part of the matrix is ignored and replaced by a transpose of the upper triangular matrix.

[scenarios, file_names, file_dir] = **parameter_set.supported_scenarios** (parse_shortnames)		
Description	Returns a list of supported scenarios.	
Input	parse_shortnames	This optional parameter can disable (0) the shortname parsing. This is significantly faster. By default shortname parsing is enabled (1).
Output	scenarios file_names file_dir	A cell array containing the scenario names. Those can be used in '`track.scenario`' The names of the configuration files for each scenario. The directory where each file was found. You can place configuration file also in you current working directory.

update_parameters (force)		
Description	Generates the LSP maps and updates the parameters for all terminals. This function calculates correlated large scale parameters for each user position. Those parameters are needed by the channel builder class to calculate initial parameters for each track or segment which are then evolved into time varying channels. By default, '`update_parameters`' reads the values given in the '`track`' objects of the '`layout`'. If there are no values given or if parts of the values are missing, the correlation maps are generated to extract the missing parameters.	
Input	force	Changes the behavior of the function. force = 0 (default) Tries to read the parameters from '`layout.track.par`'. If they are not provided or it they are incomplete, they are completed with values from the LSP maps. If the maps are invalid (e.g. because they have not been generated yet), new maps are created. force = 1 Creates new maps and reads the LSPs from those maps. Values from '`layout.track.par`' are ignored. Note that the parameters 'pg' and 'kf' will still be taken from '`layout.track.par`' when generating channel coefficients. force = 2 Creates dummy data for the maps and the LSPs. Any existing maps will be deleted. Data and maps will be declared as invalid and the next time when '`update_parameters`' is called, new parameters are generated. Values in '`layout.track.par`' will NOT be affected.

2.2.6 Class "channel_builder"

This class implements all functions that are needed to generate the channel coefficients. It thus implements the core components of the channel model. The class holds all the input variables as properties. It's main function '`get_channels`' then generates the coefficients. The procedure is summarized as follows:

The channel builder first generates a set of random clusters around each receiver. This is done by drawing random variables for the delay, the power and the departure and arrival angles for each cluster. Each cluster thus represents the origin of a reflected (and scattered) signal. The clusters are then represented as taps in the final CIR. The random variables fit the distributions and correlations defined by the '`parameter_set`' object.

Next, antenna dependent parameters are extracted for each user. Those depend on the position of the terminal, its orientation and the equipped antennas. The polarization rotation of the NLOS taps is modeled by a random variable which fits to the distribution defined by the '`parameter_set`'. The LOS polarization is calculated from the geometric orientation of the antennas. A core function here is the interpolation of the antenna patterns which results in a specific H and V value for each subpath.

The core function then generates the coefficients themselves. This is done for each antenna element and for each snapshot separately and also includes the Doppler shift of each subpath. Finally, the K-factor and the shadow fading are applied and a all the data is returned as an '`channel`' object.

Properties

name	Name of the '`channel_builder`' object
par	The '`parameter_set`' object for this channel builder
taus	The initial delays for each path in [s]. Rows correspond to the MTs, columns to the paths.
pow	The normalized initial power (squared average amplitude) for each path. Rows correspond to the MTs, columns to the paths. The sum over all columns must be 1.
AoD	The initial azimuth of departure angles for each path in [rad].
AoA	The initial azimuth of arrival angles for each path in [rad].
EoD	The initial elevation of departure angles for each path in [rad].
EoA	The initial elevation of departure angles for each path in [rad].
xpr	The initial cross polarization power ratio in [dB] for each sub-path. The dimensions correspond to the MT, the path number, and the sub-path number.
pin	The initial phases in [rad] for each sub-path.
kappa	The phase offset angle for the circular XPR in [rad]. The dimensions correspond to the MT, the path number, and the sub-path number.
random_pol	Random phasors for the WINNER polarization coupling method. The dimensions correspond to polarization matrix index '`[ 1 3 ; 2 4 ]`', the subpath number, and the MT.
subpath_coupling	A random index list for the mutual coupling of subpaths at the Tx and Rx. The dimensions correspond to the subpath index (1-20), the angle (AoD, AoA, EoD, EoA), the path number and the MT.

Methods

h_cb = **channel_builder** (h_parset)

Description	Creates a new '`channel_builder`' object.	
Input	h_parset	A '`parameter_set`' object.
Output	h_cb	A '`channel_builder`' object.

h_channel = **get_channels**

Description	Generates the channel coefficients. This is the main function of the '`channel_builder`'.	
Output	h_channel	An array of '`channel`' objects.

h_channel = **channel_builder.get_los_channels** (h_parset)		
Description	Generates channel coefficients for the LOS path only. This function generates static coefficients for the LOS path only. This includes the following properties: • antenna patterns • orientation of the Rx (if provided) • polarization rotation for the LOS path • plane-wave approximation of the phase • path loss • shadow fading No further features of QuaDRiGa are used (i.e. no drifting, spherical waves, time evolution, multipath fading etc.). This function can thus be used to acquire quick previews of the propagation conditions for a given layout.	
Input	h_parset	A '`parameter_set`' object (see Section 2.2.5).
Output	h_channel	A '`channel`' object (see Section 2.2.7). The output contains one coefficient for each position in '`h_parset.position`'.

2.2.7 Class "channel"

Objects of this class are the output of the channel model. They are created by the '`channel_builder`'. By default, channel coefficients are provided in time domain, as a list of delays and complex-valued amplitudes. However, this class also implements certain methods to postprocess the channel data. Those include:

- Transformation into frequency domain
- Interpolation in time domain (to change the terminal speed and sampling rate)
- Combining channel traces into longer segments (including birth and death of clusters)

Properties

name	Name of the '`channel`' object. This string is a unique identifier of the '`channel`' object. The '`channel_builder`' creates one channel object for each MT, each Tx and each segment. They are further grouped by scenarios (propagation environments). The string consists of four parts separated by an underscore '_'. Those are: • The scenario name from '`track.scenario`' • The transmitter name from '`layout.tx_name`' • The receiver name from '`layout.rx_name`' • The segment number After '`channel.merge`' has been called, the name string consists of: • The transmitter name from '`layout.tx_name`' • The receiver name from '`layout.rx_name`'
version	Version number of the QuaDRiGa release that was used to create the '`channel`' object.
individual_delays	Indicates if the path delays are identical on each MIMO link (0) or if each link has a different path delay (1).
coeff	The complex-valued channel coefficients for each path. The indices of the 4-D tensor are: [Rx-Antenna , Tx-Antenna , Path , Snapshot]
delay	The delays for each path. There are two different options. If the delays are identical on the MIMO links, i.e. '`individual_delays = 0`', then '`delay`' is a 2-D matrix with dimensions [Path , Snapshot]. If the delays are different on the MIMO links, then '`delay`' is a 4-D tensor with dimensions [Rx-Antenna , Tx-Antenna , Path , Snapshot].
initial_position	The snapshot number for which the initial LSPs have been generated. Normally, this is the first snapshot. However, if the user trajectory consists of more than one segment, then '`initial_position`' points to the snapshot number where the current segment starts. For example: If '`initial_position`' is 100, then snapshots 1-99 are overlapping with the previous segment.
tx_position	Position of each Tx in global cartesian coordinates using units of [m].
rx_position	The receiver position global cartesian coordinates using units of [m] for each snapshot.
no_rx no_tx no_path no_snap	Number of receive elements (read only) Number of transmit elements (read only) Number of paths (read only) Number of snapshots (read only)

Methods

h_channel = **channel** (Ccoeff, Cdelay, Cinitial_position)

Description	Creates a new '`channel`' object.	
Input	Ccoeff	The complex-valued channel coefficients for each path.
	Cdelay	The delays for each path.
	Cinitial_position	The snapshot number for which the initial LSPs have been generated.
Output	h_channel	A '`channel`' object.

freq_response = **fr**(bandwidth, carriers, i_snapshot)		
Description	Transforms the channel into frequency domain and returns the frequency response.	
Input	bandwidth	The baseband bandwidth in [Hz].
	carriers	The carrier positions. There are two options: 1. Specify the total number of carriers. In this case, '`carriers`' a scalar natural number > 0. The carriers are then equally spaced over the bandwidth. 2. Specify the pilot positions. In this case, '`carriers`' is a vector of carrier positions. The carrier positions are given relative to the bandwidth where '0' is the begin of the spectrum and '1' is the end. For example, if a 5 MHz channel should be sampled at 0, 2.5 and 5 MHz, then '`carriers`' must be set to [0, 0.5, 1].
	i_snapshot	The snapshot numbers for which the frequency response should be calculated. By default, i.e. if '`i_snapshot`' is not given, all snapshots are processed.
Output	freq_response	The complex-valued channel coefficients for each carrier in frequency domain. The indices of the 4-D tensor are: [Rx-Antenna , Tx-Antenna , Carrier-Index , Snapshot]

c = **interpolate**(dist, track_length, method)		
Description	Interpolates the channel coefficients and delays. The channel builder creates one snapshot for each position that is listed in the track object. When the channel sampling theorem is not violated (i.e. the sample density is ≥ 2), then the channel can be interpolated to any other position on the track. This can be used e.g. to emulate arbitrary movements along the track. For more information see '`track.movement_profile`', '`track.interpolate_movement`', or the tutorial "Applying Varying Speeds (Channel Interpolation)" in Section A.6.	
Input	dist	A vector containing distance values on the track. The distance is measured in [m] relative to the beginning of the track.
	track_length	The total length of the track in [m].
	method	Selects the interpolation algorithm. The default is linear interpolation. Optional are: • linear - Linear interpolation (optimized for speed) • spline - Cubic spline interpolation of the channel coefficients and piecewise cubic hermite polynomial interpolation for the delays
Output	c	A '`channel`' object containing the interpolated coefficients and delays for each entry in 'dist'.

c = **merge**(overlap, optimize, verbose)		
Description	Combines channel segments into a continuous time evolution channel. The channel merger implements the continuous time evolution with smooth transitions between segments. Each segment of a track is split in two parts: an overlapping area with the previous segment and an exclusive part with no overlapping. Each segment is generated independently by the channel builder. However, the distance dependent autocorrelation of the large scale parameters was considered when the parameters were drawn from the corresponding statistics. Transition from segment to segment is carried out by replacing taps of the previous segment by the taps of the current segment, one by one. The modeling of the birth/death process is done as published in the documentation of the WIM2 channel model. The route between adjacent channel segments is split into sub-intervals equal to the minimum number of taps in both overlapping segments. During each sub-interval the power of one old tap ramps down and one new tap ramps up. Power ramps are modeled by a modified sinus function to allow smooth transitions. Taps from the old and new segments are coupled based on their power. If the number of clusters is different in the channel segments, the weakest clusters are ramped up or down without a counterpart from the new/old segment. The merging is only done for the NLOS components since the LOS component has a deterministic behavior. The LOS component is thus just scaled in power.	
Input	overlap	The length of the overlapping part relative to the segment length. It can have values in between 0 (no overlap) and 1 (ramp along the entire segment). A value of 0 disables the merging process and the channel segments are simply concatenated. A value of 1 constantly merges the channels. The default setting is 0.5.
	optimize	The channel merger tries to automatically optimize the pairing of the taps (i.e. one tap if the old segment ramps down and one of the new ramps up). This is enabled by default, but it is computing intensive. For quicker results, it can be disabled by setting '`optimize`' to 0.
	verbose	Enables (1, default) or disables (0) the progress bar.
Output	c	An array of '`channel`' objects containing the merged coefficients and delays.

2.3 Data Flow

The data flow of the QuaDRiGa channel model is depicted in Fig. 4. This figure shows how each of the processing steps, which are described in detail in the following sections, are linked together. The lines show, which parameters are exchanged and how often they are updated. Black lines are for parameters that are either provided by the model users or which are given in the parameter table. Those values are constant. Blue values are updated once per segment and red values are updated once per snapshot.

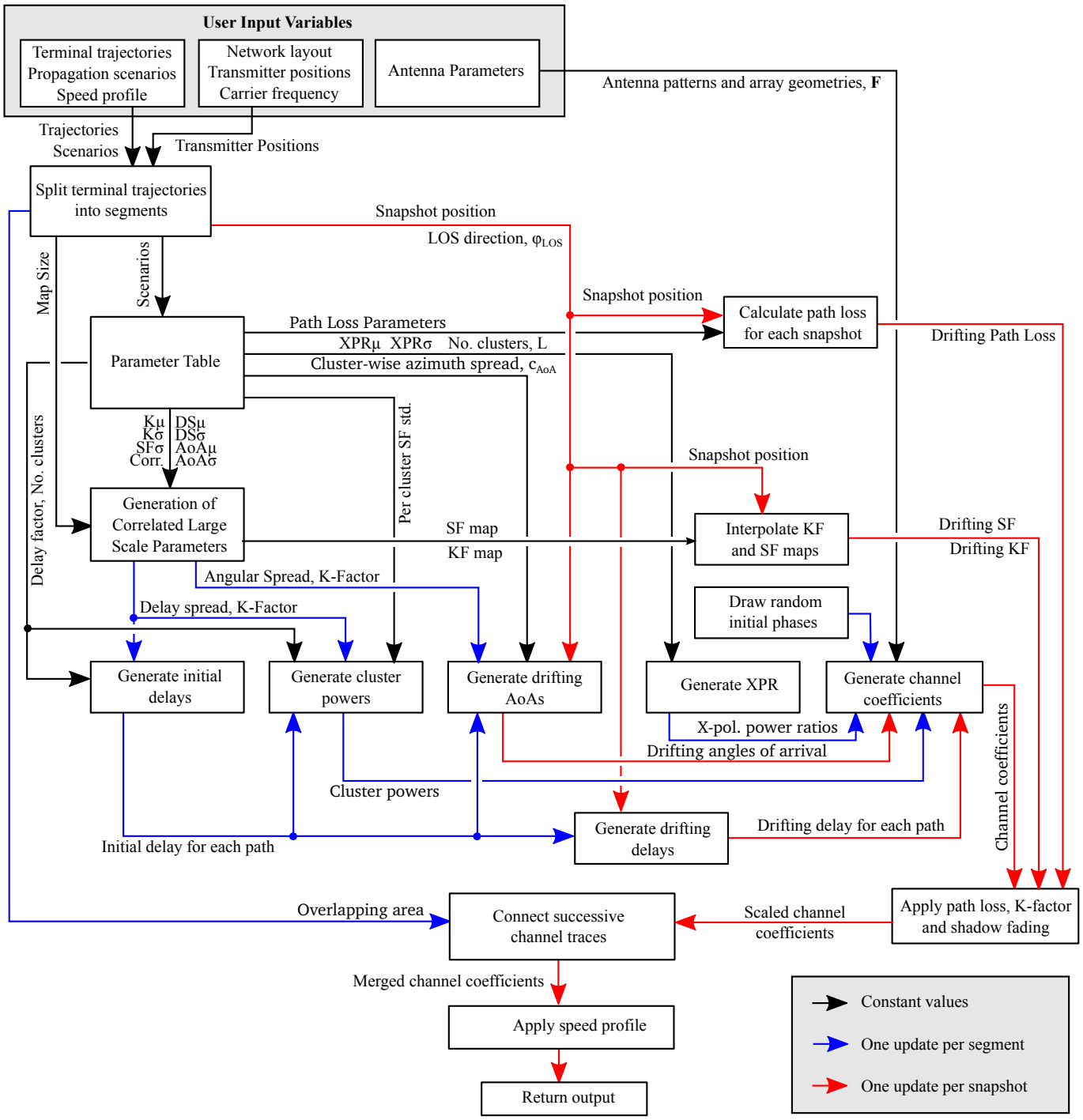

Figure 4: QuaDRiGa Data Flow

2.4 Scenario Specific Parameters

The large scale parameters (LSPs) are defined by the parameter files which can be found in the folder '`config`' of the QuaDRiGa-Core '`source`' folder. The parameters are processed as follows:

- The states and segments are identified by a name. Examples are
  `S1 = 'MIMOSA_10-45_LOS';` - parameter set selected for "good state"
  `S2 = 'MIMOSA_10-45_NLOS';` - parameter set selected for "bad state"

- The name selects the related configuration file. For the given example the files
  MIMOSA_10-45_LOS.conf and
  MIMOSA_10-45_NLOS.conf are selected

Table 9: Parameter sets provided together with the standard software

LOSonly	One LOS Path only, no Shadow-Fading, no Path-Loss
WINNER_UMa_C2_LOS WINNER_UMa_C2_NLOS	WINNER Urban Macrocell For typical terrestrial base stations deployed above rooftop in densely populated urban areas. The max. cell radius is about 1 km.
WINNER_UMi_B1_LOS WINNER_UMi_B1_NLOS	WINNER Urban Microcell For typical terrestrial pico-base stations deployed below rooftop in densely populated urban areas. The max. cell radius is about 200 m.
WINNER_SMa_C1_LOS WINNER_SMa_C1_NLOS	WINNER Sub-Urban Macrocell For typical terrestrial base stations deployed above rooftop in sub-urban areas. The max. cell radius is about 10 km.
WINNER_Indoor_A1_LOS WINNER_Indoor_A1_NLOS	WINNER Indoor Hotspot For typical indoor deployments such as WiFi or femto-cells.
WINNER_UMa2Indoor_C4_LOS WINNER_UMa2Indoor_C4_NLOS	WINNER Urban Macrocell to Indoor For users within buildings that are connected to a terrestrial base station deployed above rooftop in densely populated urban areas.
WINNER_UMi2Indoor_B4_LOS WINNER_UMi2Indoor_B4_NLOS	WINNER Urban Microcell to Indoor For users within buildings that are connected to terrestrial pico-base stations deployed below rooftop in densely populated urban areas.
BERLIN_UMa_LOS BERLIN_UMa_NLOS	Terrestrial Urban Macrocell parameters extracted from measurements in Berlin, Germany.
MIMOSA_10-45_LOS MIMOSA_10-45_NLOS	MIMOSA Satellite to Mobile Parameters for Urban Propagation Elevation range from 10 to 45°. Parameters were extracted from terrestrial measurement using a high-attitude platform.
MIMOSA_16-25_LOS MIMOSA_16-25_NLOS	MIMOSA Satellite to Mobile Parameters for Urban Propagation Elevation range from 16 to 25°. Parameters were extracted from terrestrial measurement using a high-attitude platform.
MIMOSA_25-35_LOS MIMOSA_25-35_NLOS	MIMOSA Satellite to Mobile Parameters for Urban Propagation Elevation range from 25 to 35°. Parameters were extracted from terrestrial measurement using a high-attitude platform.
MIMOSA_35-45_LOS MIMOSA_35-45_NLOS	MIMOSA Satellite to Mobile Parameters for Urban Propagation Elevation range from 35 to 45°. Parameters were extracted from terrestrial measurement using a high-attitude platform.

2.4.1 Description of the Parameter Table

The QuaDRiGa channel model is a generic model. That means, that it uses the same method for generating channel coefficients in different environments. E.g. the principal approach is exactly the same in a cellular network and in a satellite network. The only difference is the parametrization for both cases. Each environment is described by 55 individual parameters. These parameters are stored in configuration files that can be found in the subfolder named "config" in the main channel model folder. The parameters and values can be describes as follows:

- **No. Clusters**
  This value describes the number of clusters. Each cluster is the source of a reflected or scattered wave arriving at the receiver. Typically there are less clusters in a LOS scenario then in a NLOS scenario. Note that the number of clusters directly influences the time needed to calculate the coefficients.

- **Path Loss (PL)**
  A common path loss (PL) model for cellular systems is the log-distance model

$$\mathrm{PL}_{[\mathrm{dB}]} = A \cdot \log_{10} d + B \tag{1}$$

  where A and B are scenario-specific coefficients. The path-loss exponent A typically varies between 2 and 4, depending on the propagation conditions, the base station height and other influences. They are typically determined by measurements. d is the distance (in units of meters) between the transmitter and the receiver. Note that in Tab. **??**, a factor of 10 is applied to the PL-exponent. In other environments such as in satellite systems, the PL does not depend on the distance but has a constant value. In this case, A would be 0.

- **Shadow Fading (SF)**
  Shadow fading occurs when an obstacle gets positioned between the wireless device and the signal transmitter. This interference causes significant reduction in signal strength because the wave is shadowed or blocked by the obstacle. It is modeled as log-normal distributed with two parameters: The standard deviation σ defines the width of the distribution, i.e. the power value (in dB) above or below the distance dependent PL and the decorrelation distance λ. This parameter defines how fast the SF varies when the terminal moves through the environment. E.g. a value of 87 means that when the terminal moves 87 m in any given direction, then the correlation of the value at this distance with the value at the initial position is $e^{-1} = 0.37$.

- **Delay Spread (DS)**
  The root-mean-square (RMS) delay spread is probably the most important single measure for the delay time extent of a multipath radio channel. The RMS delay spread is the square root of the second central moment of the power delay profile and is defined to be

$$\sigma_\tau = \sqrt{\frac{1}{P_i} \cdot \sum_{l=1}^{L} P_l \cdot (\tau_l)^2 - \left(\frac{1}{P_i} \cdot \sum_{l=1}^{L} P_l \cdot \tau_l \right)^2} \tag{2}$$

  with P_i is the total received power, P_l the cluster-power and τ_l the cluster delay.

  In order to generate the coefficients, QuaDRiGa has to generate delays for each of the multipath clusters. I.e. the total lengths of scattered paths have to be defined. This generation of delays is governed by value of the DS in a specific environment. The DS is assumed to be log-normal distributed and defined by two parameters: Its median value μ and its STD σ. Thus, a values of DS_μ of -6.69 corresponds to 204 ns. σ then defines the range of possible values. E.g. $\mathrm{DS}_\sigma = 0.3$ leads to typical values in the range of $10^{-6.69-0.3} = 102$ ns to $10^{-6.69+0.3} = 407$ ns. As for the shadow fading, the decorrelation distance DS_λ defines how fast the DS varies when the terminal moves through the environment.

  The delay spread σ_τ is calculated from both, the delays τ_l and the path powers P_l. I.e. lager delay spreads σ_τ can either be achieved by increasing the values of τ_l and keeping P_l fixed or adjusting P_l and keeping τ_l fixed. In order to avoid this ambiguity, an additional proportionality factor (delay factor) r_τ is introduced to scale the width of the distribution of τ_l. r_τ is calculated from measurement data. See Sec. 3.2.1 for more details.

- **Rician K-Factor (KF)**
  Rician fading occurs when one of the paths, typically a line of sight signal, is much stronger than

the others. The KF K is the ratio between the power in the direct path and the power in the other, scattered, paths. As for the DS, the KF is assumed to be log-normal distributed. The distribution is defined by its median value KF_μ and its STD KF_σ. The decorrelation distance KF_λ defines how fast the KF varies when the terminal moves through the environment.

- **Angular Spread**
  The angular spread defines the distribution of the departure- and arrival angles of each multipath component in 3D space seen by the transmitter and receiver, respectively. Each path gets assigned an azimuth angle in the horizontal plane and an elevation angle in the vertical plane. Thus we have four values for the angular spread:

  1. Azimuth spread of Departure (AsD)
  2. Azimuth spread of Arrival (AsA)
  3. Elevation spread of Departure (EsD)
  4. Elevation spread of Arrival (EsA)

  Each one of them is assumed to be log-normal distributed. Hence, we need the parameters μ, σ and λ to define the distributions. These spreads are translated into specific angles for each multipath cluster. Additionally, we assume that clusters are the source of several multipath components that are not resolvable in the delay domain. Thus, these sub-paths do not have specific delays, but they have different departure- and arrival angles. Thus, we need an additional parameter c_ϕ for each of the four angles that scales the dimensions of the clusters in 3D-space. See Sec. 3.2.2 for details.

- **Cross-polarization Ratio (XPR)**
  The XPR defines how the polarization changes for a multipath component. I.e. the initial polarization of a path is defined by the transmit antenna. However, for the NLOS components, the transmitted signal undergoes some diffraction, reflection or scattering before reaching the receiver.

  The XPR (in dB) is assumed to be normal distributed where μ and σ define the distribution. We translate the XPR in a polarization rotation angle which turns the polarization direction. A XPR value where a value of +Inf means that the axis remains the same for all NLOS paths. I.e. vertically polarized waves remain vertically polarized after scattering. On the other hand, a value of -Inf dB means that the polarization is turned by 90°. In case of 0 dB, the axis is turned by 45°, i.e. the power of a vertically polarized wave is split equally into a H- and V component.

The following table gives an overview of the parameters in the config files. They get converted into a structure '`parameter_set.scenpar`'.

Parameter	Unit or type	Description
plpar.model (PL_model)	Text string	Selects the model for average path loss. For satellite applications the pathloss is defined by the satellite distance and is assumed to be constant for the reception are. For terrestrial cases pathloss models like "Hata" or others (e.g. WINNER pathloss models) can be selected.
plpar.A (PL_A)	dB	For satellite applications this parameter defines the average path loss and is equivalent to the "mu" of the lognormal distribution of the shadow fading.
Parameters in structure '`parameter_set.scenpar`'		
Large-Scale Parameters		Those parameters describe how the large-scale parameters vary within a propagation environment.
SF_sigma SF_lambda	dB meter	Those parameter describe the slow fading implemented as Lognormal distribution and filtered (see "map" generation) according to the de-correlation distance "lambda".
KF_mu KF_sigma KF_lambda	dB dB meter	Statistical properties of the K-factor

xpr_mu xpr_sigma	dB dB	The XPR is defined by the XPR-Antenna (see antenna pattern) and the XPR "environment". The parameters describe the statistical properties of the XPR "environment". For the XPR, no correlation map is calculated and the XPR is updated once per segment. Note: For the LOS component no XPR environment is assumed, only the XPR antenna is applied. Hence the overall XPR depends also highly on the K-factor.
DS_mu DS_sigma DS_lambda	log10([s]) log10([s]) meter	Statistical properties of the delay spread.
AS_A_mu AS_A_sigma AS_A_lambda	log10([deg]) log10([deg]) meter	Statistical properties of the azimuth of arrival spread at the receiver.
ES_A_mu ES_A_sigma ES_A_lambda	log10([deg]) log10([deg]) meter	Statistical properties of the elevation of arrival spread at the receiver.
AS_D_mu AS_D_sigma AS_D_lambda	log10([deg]) log10([deg]) meter	Statistical properties of the azimuth of departure spread at the transmitter.
ES_D_mu ES_D_sigma ES_D_lambda	log10([deg]) log10([deg]) meter	Statistical properties of the elevation of departure spread at the transmitter.
Cross correlations		There are interdependencies between parameters. For example, if the K-Factor is high, the delay spread gets shorter since more power is coming from the direct component. This is expressed by the cross-correlations parameters. They can vary between -1 and 1. Negative values denote inverse correlation, e.g. a high K-Factor implies a low delay spread. Positive Value implies a positive correlation such as a high K-Factor also implies a high shadow fading. Cross-Correlations are used during the map-generation.
ds_kf asA_ds esA_ds ds_sf asA_kf esA_kf sf_kf esA_asA asA_sf esA_sf		Correlation of delay spread and K-Factor. Correlation of delay spread and azimuth of arrival angle spread. Correlation of delay spread and elevation of arrival angle spread. Correlation of delay spread and shadow fading. Correlation of K-Factor and azimuth of arrival angle spread. Correlation of K-Factor and elevation of arrival angle spread. Correlation of K-Factor and shadow fading. Correlation elevation of arrival angle spread and azimuth of arrival angle spread. Correlation of shadow fading and azimuth of arrival angle spread. Correlation of shadow fading and elevation of arrival angle spread.
Cluster Parameter		Those parameters influence the generation of the scattering-clusters and the distribution of the sub-paths within each cluster.
NumClusters	Integer	The number of clusters generated. For multipath rich environments typically more clusters are used. If the LOS component is dominant a lower number of clusters is sufficient.
PerCluster AS_A	deg	The azimuth angular spread of the 20 sub-paths within one cluster.
PerCluster ES_A	deg	The elevation angular spread of the 20 sub-paths within one cluster.
LOS_scatter_radius	meter	This parameter allows an additional spread of the 20 sub-paths of the LOS component by emulating scattering in the near-field of the antennas. [EXPERIMENTAL]
LNS_ksi		Normally, cluster powers are taken from an exponential power-delay-profile. This parameter enables an additional variation of the individual cluster powers around the PDP.
r_DS		This parameter allows the mapping of delay-spreads to delays and powers for the clusters. See section 3.2.1.

2.4.2 Adding New Scenarios

The scenario parameters are set in "parameter_set.scenpar". Here, you also have the option to change individual parameters by assigning new values. The scenario "UMal", for example, uses by default 8 clusters. When the simulations should be done with 15 clusters, one can change the settings by

```
p = parameter_set('BERLIN_UMa_LOS');          % New parameter set
p.scenpar.NumClusters = 15;                   % Set new number of clusters
```

A list of currently supported scenarios is generated by:

```
parameter_set.supported_scenarios
```

The default settings of those scenarios are stored in config files which are located in the "config"-folder of the QuaDRiGa source path. The "UMal" config file for example looks like this:

```
% Config File for scenario "UMal"
% WINNER+ Urban Macro LOS
% See: CELTIC / CP5-026 D5.3: WINNER+ Final Channel Models
% and: IST-4-027756 WINNER II D1.1.2 v.1.1: WINNER II Channel Models
%
% Stephan Jaeckel
% Fraunhofer Heinrich Hertz Institute
% Wireless Communication and Networks
% Einsteinufer 37, 10587 Berlin, Germany
% e-mail: stephan.jaeckel@hhi.fraunhofer.de

NumClusters = 8
r_DS        = 2.5

SF_sigma    = 5
SF_lambda   = 45
LNS_ksi     = 3

KF_mu       = 7
KF_sigma    = 3
KF_lambda   = 12

DS_mu       = -7.39
DS_sigma    = 0.63
DS_lambda   = 40

AS_D_mu     = 1
AS_D_sigma  = 0.25
AS_D_lambda = 15
PerClusterAS_D = 6

AS_A_mu     = 1.7
AS_A_sigma  = 0.19
AS_A_lambda = 15
PerClusterAS_A = 12

ES_D_mu     = 0.7
ES_D_sigma  = 0.2
ES_D_lambda = 15    % D5.3 pp. 73
PerClusterES_D = 3

ES_A_mu     = 0.95
ES_A_sigma  = 0.16
ES_A_lambda = 15    % D5.3 pp. 73
PerClusterES_A = 7

xpr_mu      = 8
xpr_sigma   = 4

% Adjustments have been made to keep xcorr-matrix positive definite
asD_ds  =  0.4
asA_ds  =  0.7 % 0.8
asA_sf  = -0.5
asD_sf  = -0.4 % 0.5
```

```
ds_sf    = -0.4
asD_asA =  0.3
asD_kf   =  0.1
asA_kf   = -0.2
ds_kf    = -0.4
sf_kf    =  0.3
esD_ds   = -0.4 % -0.5
esD_asD =  0.4 % 0.5
esA_sf   = -0.8
esA_asA =  0.4

% A * log10 d + B + C * log10 f + D * log10 hBS + E * log10 hMS
% Two different values (first before breakpoint, last after breakpoint)
% Different SF coefficients

PL_model = winner_los

PL_A1    = 26
PL_B1    = 25
PL_C1    = 20
PL_D1    = 0
PL_E1    = 0
PL_sig1 = 4

PL_A2    = 40
PL_B2    = 9.27
PL_C2    = 6
PL_D2    = -14
PL_E2    = -14
PL_sig2 = 6
```

You can create you own scenario by editing this file and saving it under a new filename in the "config"-Folder. The file ending must be ".conf". The filename then is also the scenario name and the settings can be accessed from inside MATLAB as described above.

3 Technical Documentation

An overview of the modeling steps is given in Fig. 5. The user provides the network layout, i.e. the positions of the BSs, antenna configurations, downtilts, the positions and trajectories of the MTs and the propagation scenarios. The channel coefficients are then calculated in seven steps which are described in sections 2.4 and 3.2.

Much of the modeling approach is inspired by the WINNER model [KMH+07]. Major extensions concerning the time evolution have been made in steps D and G. In order to make those extension work properly, changes were also required in the other parts. For example, time evolution requires a more detailed description of the mobility of the terminals which is solved by assigning tracks, i.e. ordered lists of positions, to the MTs. Updates of the channel coefficients are then triggered at fixed snapshot positions.

The WINNER model only allows constant speeds via a continuous rotation of the phases of the MPCs. However, realistic scenarios would also include accelerations, decelerations and MTs with different speeds, e.g. pedestrian and vehicular users. However, to minimize the computational overhead and memory requirements, we generate the channel coefficients at a constant sample rate which fulfills the sampling theorem, i.e. updates are triggered at least twice per half wave length. A time-series for varying speeds is then obtained by interpolating the coefficients in a separate postprocessing step.

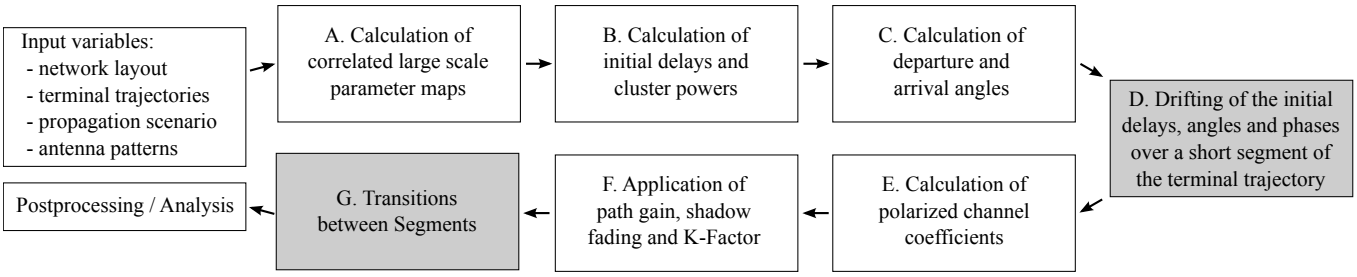

Figure 5: Essential steps for the calculation of time evolving channel coefficients

3.1 Tracks, Scenarios, Antennas and Network Layout

3.1.1 Drops and Segments

The concept of segments and drops is already described in [KMH+07]. See also Fig. 6. Channel segments represent a period of quasi-stationarity during which probability distributions of low-level parameters are not changed noticeably. During this period all large-scale parameters are practically constant. To be physically feasible, the channel segment must be confined in distance. The size of the segments depends on the environment, but it can be at maximum a few dozen meters. The decorrelation distances of different parameters describe roughly the proper size of the channel segment. These decorrelation distances can be extracted from measurements and are scenario dependent.

In the WINNER terminology, a drop is defined as a segment with zero-length. In order to extend the length of a drop, short-term time-variability of some channel parameters is added within the drops. This method is known as drifting and was first described in [BHS05]. The current model uses drifting to enable time continuous simulations. Here, two types of drifting need to distinguished:

1. Drifting of Path Delays and Angles and Polarization
   It is assumed that the positions of scatterers are fixed during a drop. As a consequence, the angle of departures (AoDs) of the scatters as seen from the BS do not change, with the exception of the LOS AoD. However, based on the fixed-geometry assumption, the angle of arrivals (AoAs) of the scatters

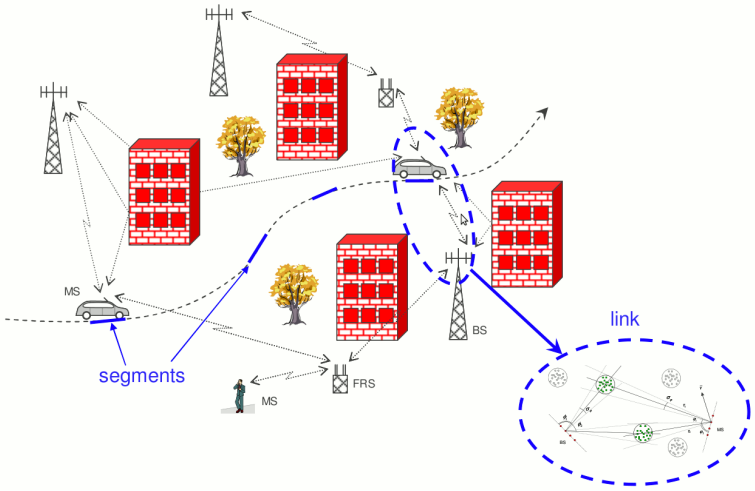

Figure 6: WINNER system level approach showing several segments (drops). Source: [KMH+07]

as seen from the MT as well as the subpath delays change during a drop due to the MT movement. Similarly, the LOS directions between BS and MT and the polarization characteristics vary in time. Implementation details of the drifting can be found in Section 3.2.3.

2. Drifting of Shadow Fading and K-Factor
   The time-evolution of shadow fading is determined by its spatial autocorrelation function. References show that an exponential function fits well and the drifting can thus be modeled by a first order autoregressive process. This is realized by calculating maps containing the correlated large scale parameters. The drifting is then implemented by reading the map values along the segments track and interpolating the data.

3.1.2 Sample Density vs. Sample Rate

If you consider moving receivers, choosing the right sampling rate depends directly on the Doppler spectrum. In order to successfully reconstruct the impulse response, you need a sample rate f_T fulfilling the sampling theorem.

$$f_T \geq 2B_D = 4 \max |\Delta f_D| = 4 \frac{\max |v|}{\lambda_c} \tag{3}$$

with B_D being the width of the Doppler spectrum, Δf_D the frequency change due to velocity v and λ_c being the carrier wavelength. Thus the appropriate sampling rate is proportional to the maximal velocity of the receivers. Since it is sometimes useful to examine algorithm behavior for different user velocities, it is unfortunate to fix the sampling rate in advance as the max speed is fixed as well. To overcome this problem it is advantageous to sample the channel in the spatial domain rather than in the time domain.

$$f_S = \frac{f_T}{\max |v|} \geq \frac{4}{\lambda_c} \tag{4}$$

Here f_S denotes the spatial sampling rate in samples per meter. In its normalized form it is also known as sample density SD

$$\mathrm{SD} = \frac{f_S}{\lambda_c / 2} \geq 2 \tag{5}$$

Another advantage of sampling in the spatial domain is the reduction of required storage, since the number of samples is reduced in case the maximum speed is greater than 1 meter per second (3.6 kilometers per hour).

3.1.3 The Antenna Model

One feature of `QuaDRiGa` is that it allows the separations of propagation- and antenna effects. An antenna is defined by its directional response F, also known as field pattern. The original SCM considers only 2D propagation. Thus, F is a function of the azimuth angle ϕ which is also indicated in Fig. 2. Later extensions [SZM+06] also consider the elevation direction θ. The complex amplitude g of one tap between a transmit antenna t and a receive antenna r notes

$$g_{r,t} = \sqrt{P} \cdot \mathbf{F}_r(\phi^a, \theta^a)^T \cdot \mathbf{M} \cdot \mathbf{F}_t(\phi^d, \theta^d) \cdot e^{-j\frac{2\pi}{\lambda} \cdot d} \tag{6}$$

where $\mathbf{F}_r$ and $\mathbf{F}_t$ are the patterns at the receiver and transmitter, respectively. λ is the wavelength and P is the power of the tap. Note that the patterns $\mathbf{F}(\phi, \theta)$ are two-element vectors which contain the vertically (F_V) and horizontally (F_H) polarized component of the antenna response

$$\mathbf{F}(\phi, \theta) = \begin{pmatrix} F_V(\phi, \theta) \\ F_H(\phi, \theta) \end{pmatrix} \tag{7}$$

$\mathbf{M}$ is the 2×2 polarization coupling matrix. This matrix describes how the polarization changes on the way from transmitter to receiver. It is important to note that in a MIMO configuration, which uses multiple antennas at transmitter and receiver, the physical propagation effects such as the angles of departure and arrival, the path powers, the delays and the polarization coupling stay the same for all antennas. The only difference is that each antenna element has a different field pattern.

Changing the orientation of antennas Here we describe how the orientation of an antenna can be changed (e.g. when moving the receiver along a street, or turning a mobile phone in the hands of the user). This is done in two steps. First, the field patterns for both polarizations F_{V} and F_{H} are rotated and interpolated separately. The second step then includes the effects of the polarization.

An antenna pattern (7) is given in spherical coordinates as a function of the azimuth angle ϕ and elevation angle θ. When the orientation of the antenna element changes, the field pattern has to be read at different

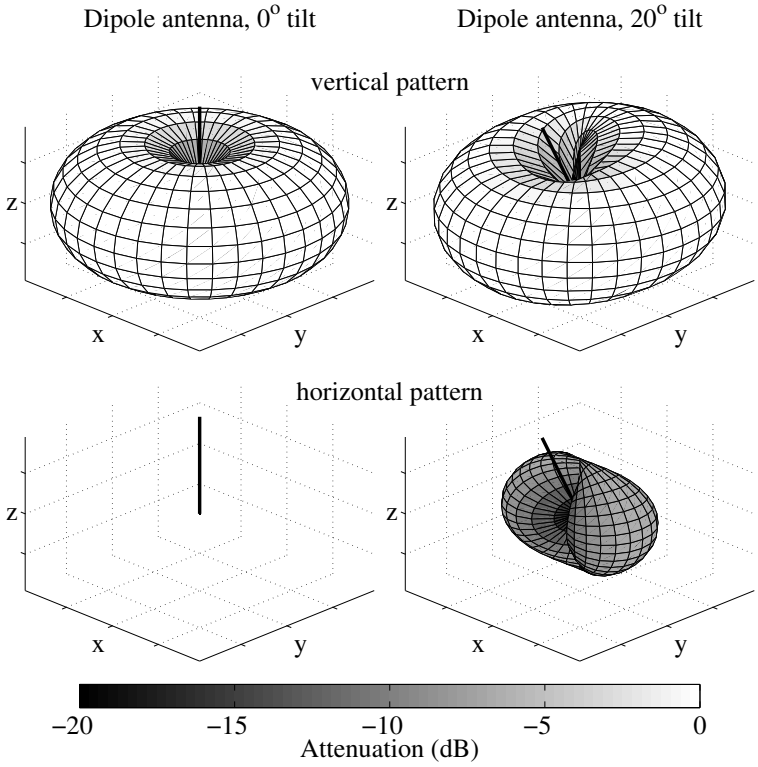

Figure 7: Example patterns for a dipole antenna.

angles (Φ, Θ) which include the effect of the orientation change. Rotations in 3D are easier in Cartesian coordinates. We therefore transform the original angle pair (ϕ, θ) into a vector $\mathbf{a}$ that describes the arrival- or departure angles in Cartesian coordinates. The three vector elements represent the x,y and z-component.

$$\mathbf{a}(\phi, \theta) = \begin{pmatrix} \cos\phi \cdot \cos\theta \\ \sin\phi \cdot \cos\theta \\ \sin\theta \end{pmatrix} \tag{8}$$

We now use a 3×3 matrix $\mathbf{Q}$ to describe the orientation change in 3D space. In principal, we can calculate $\mathbf{Q}^T$ for any arbitrary rotation axis and angle. The example in Fig. 7 was tilted by 20° around the x-axis of the coordinate system. The corresponding matrix is

$$\mathbf{Q}_x(20^\circ) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(20^\circ) & -\sin(20^\circ) \\ 0 & \sin(20^\circ) & \cos(20^\circ) \end{pmatrix} \tag{9}$$

By multiplying $\mathbf{Q}$ with (8), we include the orientation change in the vector

$$\mathbf{a}^+(\phi, \theta) = \mathbf{Q}^T \cdot \mathbf{a}(\phi, \theta) \tag{10}$$

Since $\mathbf{a}^+$ is now also in Cartesian coordinates and we need the transformed pattern $\tilde{\mathbf{F}}$ in spherical coordinates, we have to transform $\mathbf{a}^+$ back to spherical coordinates. This results in the new angles (Φ, Θ).

$$\Phi(\phi, \theta) = \arctan_2\left[a_y^+(\phi, \theta), a_x^+(\phi, \theta)\right] \tag{11}$$

$$\Theta(\phi, \theta) = \arcsin\left[a_z^+(\phi, \theta)\right] \tag{12}$$

a_x^+, a_y^+ and a_z^+ are the x, y and z component of $\mathbf{a}^+$, respectively. We now get the coefficients of the rotated pattern by reading the original pattern $\mathbf{F}$ at the transformed angles

$$\hat{F}_{\mathrm{V/H}}(\phi, \theta) = F_{\mathrm{V/H}}(\Phi, \Theta) \tag{13}$$

Since the patterns are sampled at a fixed angular grid, this involves an interpolation step. As a standard computationally inexpensive procedure, we use linear interpolation. Alternatively, more advanced techniques based on the effective aperture distribution function (EADF) can be used [NKS+07].

The second step of the transformation takes the polarization into account. We first take the antenna orientation vector $\mathbf{o}$ and apply the rotation matrix $\mathbf{Q}$ to obtain $\tilde{\mathbf{o}}$

$$\tilde{\mathbf{o}} = \mathbf{Q} \cdot \mathbf{o} \tag{14}$$

This vector $\tilde{\mathbf{o}}$ is the new orientation vector of the transformed pattern. Next, we calculate a rotation matrix that accounts for the changed polarization characteristics of the rotated antenna pattern. In principal, the following procedure emulates an antenna measurement like in an anechoic chamber. The virtual transmitter is placed south of our test antenna. The wave travel direction $\mathbf{r}$ thus lies in y direction. We thus rotate the orientation vector of our antenna so that it matches the wave travel direction. The polarization vector $\mathbf{p}_r$ results from a projection of the orientation vector $\mathbf{o}_r$ on the plane perpendicular to the travel direction (see Fig. 13 for details).

1. We rotate the orientation vector $\tilde{\mathbf{o}}$ by $p = -\phi - \pi/2$ in azimuth direction and $q = \theta$ in elevation direction to match the orientation of the transmitter.

$$\mathbf{o}^+ = \begin{pmatrix} \cos p & -\sin p & 0 \\ \cos q \cdot \sin p & \cos q \cdot \cos p & -\sin q \\ \sin q \cdot \sin p & \sin q \cdot \cos p & \cos q \end{pmatrix} \cdot \tilde{\mathbf{o}} \tag{15}$$

2. We calculate the projection of the orientation vector on the projection plane. Since the projection plane lies in the x-z-plane due to the placement of the transmitter, we simply omit the y component of $\mathbf{o}^+$, switch

the x and z component and normalize the resulting vector to unit length. The switching is done to obtain the same orientation as in (7).

$$\mathbf{p}_r = \begin{pmatrix} o_z^+ \\ o_x^+ \end{pmatrix} / \left| \begin{pmatrix} o_z^+ \\ o_x^+ \end{pmatrix} \right| \tag{16}$$

3. We define the transmitter to be either perfectly vertical $\left(\mathbf{p}_t = [1,0]^T\right)$ or perfectly horizontal $\left(\mathbf{p}_t = [0,1]^T\right)$. As a consequence, the product $\mathbf{p}_r^T \cdot \mathbf{p}_t$ selects either the vertical or horizontal component of $\mathbf{p}_r$ and α can be calculated to

$$\alpha = \arctan_2\left(p_{ry}, p_{rx}\right) = \arctan_2\left(o_x^+, o_z^+\right) \tag{17}$$

4. α is the angle between the projection of the orientation vector and the E_x component of the transmit polarization. Since the transmitter is vertically polarized, β_t is 0. The angle β_r comes from the rotated field pattern response (13).

$$\beta_r = \arctan_2\left\{\hat{F}_H(\phi,\theta), \hat{F}_V(\phi,\theta)\right\} \tag{18}$$

5. The difference between α and β_r is the rotation angle ϑ which is used to calculate the polarization effects on the pattern. The rotated pattern then notes

$$\begin{aligned} \vartheta &= -\beta_r - \alpha \\ \tilde{\mathbf{F}}(\phi,\theta) &= \begin{pmatrix} \cos\vartheta & -\sin\vartheta \\ \sin\vartheta & \cos\vartheta \end{pmatrix} \cdot \begin{pmatrix} \hat{F}_{\mathrm{V}}(\phi,\theta) \\ \hat{F}_{\mathrm{H}}(\phi,\theta) \end{pmatrix} \end{aligned} \tag{19}$$

Modeling circularly polarized antennas In many applications, such as satellite communications, circular polarization is needed. A straight forward extension would use complex coefficients in the field patterns where

$$\mathbf{F}_{\mathrm{R}} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ i \end{pmatrix} \quad \mathbf{F}_{\mathrm{L}} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -i \end{pmatrix} \tag{20}$$

are the Jones vectors for the RHCP and LHCP signal, respectively. However, an essential component of the polarization model is the orientation vector $\mathbf{o}$ which is not unique for circular polarized patterns. A solutions is to virtually assemble a circular polarized antenna out of two linear elements. These elements need to be crossed in a way that the transmission axes of both elements are perpendicular to each other. Both elements are fed with the same signal, but one of them is shifted by 90° out of phase. This is modeled by using the channel coefficients (6) for each of the two elements and assembling them in a channel matrix $\mathbf{G}$. In order to get circular polarized coefficient matrix $\mathbf{G}^\circ$, we use the Jones vectors (20) as coupling matrices.

$$\begin{aligned} \mathbf{G}^\circ &= \begin{pmatrix} g_{\mathrm{RR}} & g_{\mathrm{RL}} \\ g_{\mathrm{LR}} & g_{\mathrm{LL}} \end{pmatrix} = \mathbf{C}^H \mathbf{G} \mathbf{C} \\ &= \frac{1}{2} \begin{pmatrix} 1 & -i \\ 1 & i \end{pmatrix} \begin{pmatrix} g_{\mathrm{VV}} & g_{\mathrm{VH}} \\ g_{\mathrm{HV}} & g_{\mathrm{HH}} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ i & -i \end{pmatrix} \end{aligned} \tag{21}$$

3.1.4 Generation of Correlated Large Scale Parameters

QuaDRiGa models the auto- and cross correlation properties of the LSPs by creating 2D maps for each of the following parameters.

1. RMS Delay Spread (DS)
2. Ricean K-Factor (KF)
3. Shadow Fading (SF)
4. Azimuth spread of Departure (AsD)
5. Azimuth spread of Arrival (AsA)
6. Elevation spread of Departure (EsD)
7. Elevation spread of Arrival (EsA)

The map-based method is already part of the WINNER implementation [KMH+07, HMK+10] where the maps are generated by filtering random, normal distributed numbers along the x and y axis of the map. Alternative approaches [CG03, Cla05] are known to have better autocorrelation properties at close distances. I.e. they are better in modeling the distance-dependent exponential decay of the correlation. However, the resulting map is difficult to interpolate since neighboring pixels can have large differences in magnitude. We thus extend the WINNER idea by filtering the maps also in the diagonal directions. This significantly increases the smoothness of the parameters along a random user trajectory which is an important feature for the time evolution of the channel coefficients. The principle of the map based correlation procedure is shown in Fig. 8 for an example using the delay spread (DS) in an urban cellular scenario. The granularity of each parameter can be described on three levels: the scenario level, the link level and the path level.

Scenario Level The magnitude, variance and the correlation of a parameter in a specific scenario (e.g. urban macrocell, indoor hotspot or urban satellite) are calculated from measurement data. Normally, parameters are assumed to be log-normal distributed. For example, the median log-normal delay spread DS_μ in an urban cellular scenario is -6.89 which corresponds to a DS of $\bar{\sigma}_\tau = 128$ ns (see Fig. 8, top). With a standard deviation of $\mathrm{DS}_\sigma = 0.5$, typical values may lie in between 40 and 407 ns. The same principle applies for the other six parameters. The decorrelation distance (e.g. $\mathrm{DS}_\lambda = 40$ m) describes the distance-dependent correlation of the LSP. If e.g. two mobile terminals in the above example are 40 m apart of each other, their DS is correlated with a correlation coefficient of $e^{-1} = 0.37$. Additionally, all parameters are cross correlated. A typical example is the dependance of the angular spread (e.g. the azimuth spread of arrival) on the Ricean K-factor (KF). With a large KF (e.g. 10 dB), a significant amount of energy comes from a single direction. Thus, the angular spread gets smaller which leads to a negative correlation between the DS and the KF.

Link Level When a user terminal is placed in a scenario (black dot on the map in Fig. 8), it experiences a radio channel which is determined by the specific values of the seven parameters at this position. Due to the autocorrelation properties, small distances between users also lead to high correlations in the channel

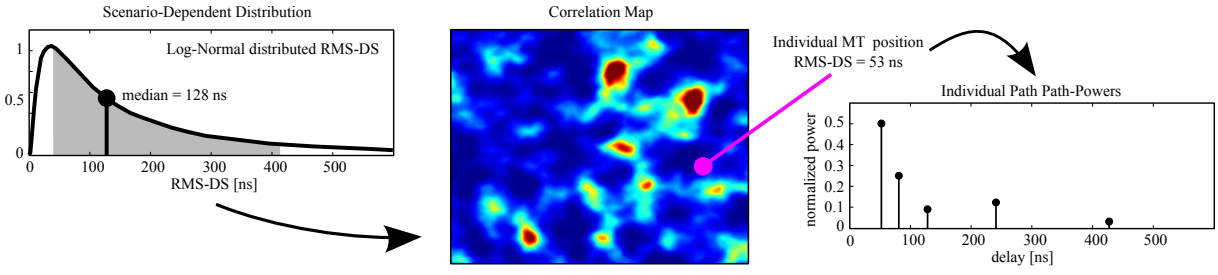

Figure 8: Principle of the generation of correlated LSPs

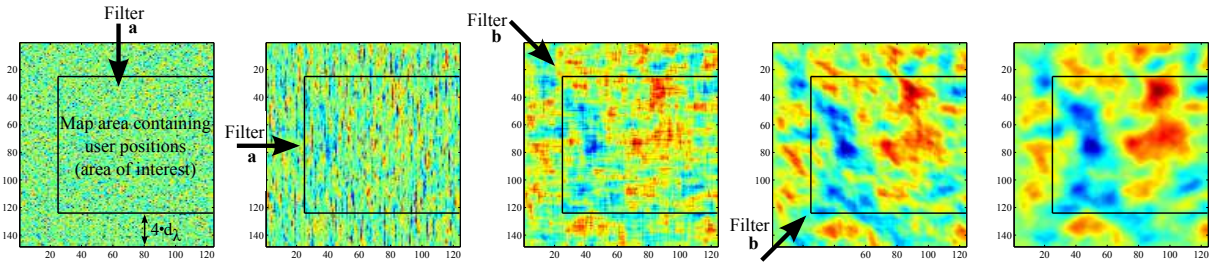

Figure 9: Principle of the map generation

statistics (e.g. a second terminal right next to the current user will experience a similar DS). The second granularity of the large scale parameters are thus the specific values of the LSPs for each user in the scenario. Generating those values can be seen as going backwards from the distribution μ, σ of a parameter to individual "measurement"-values. At the example position in the map, the DS is 53 ns.

Path Level Last, the individual components of the CIR are calculated. This procedure takes the values of the LSPs into account and calculates the power and the delay of the channel coefficients.

The correlation maps are generated at a fixed sampling grid by successively filtering a random, normal distributed sequence of numbers with a finite impulse response (FIR) filter. The principle is depicted in Fig. 9. The map is represented by a matrix $\mathbf{B}$ and one pixel of that matrix is $B_{y,x}$ where y is the row index and x is the column index. The first pixel $B_{1,1}$ is in the top left (or north-west) corner of the map. The FIR filter coefficients are calculated from the decorrelation distance d_λ (in units of m). The distance dependent correlation coefficient follows an exponential function

$$\rho(d) = e^{-\frac{d}{d_\lambda}} \tag{22}$$

with d as the distance between two positions [Gud91]. We now calculate two sets of filter coefficients, one for the horizontal and vertical directions and one for the diagonal elements. This is done by taking (22) and substituting the distance d by the relative distance d_{px} of two pixels.

$$a_k = \frac{1}{\sqrt{d_\lambda}} \cdot \exp\left(-\frac{k \cdot d_{\mathrm{px}}}{d_\lambda}\right) \tag{23}$$

$$b_k = \frac{1}{\sqrt{d_\lambda}} \cdot \exp\left(-\frac{k \cdot \sqrt{2} d_{\mathrm{px}}}{d_\lambda}\right) \tag{24}$$

k is the running filter coefficient index. We cut the exponential decay function a at maximum distance of $4d_\lambda$ and normalize it with $\sqrt{d_\lambda}$. The map size is determined by the distribution of the users in the scenario plus the length of the filter function. This is also illustrated in Fig. 9 where the user terminals are placed inside the black square. The extension space is needed to avoid filter artifacts at the edges of the map. We initialize the map with random, normal distributed numbers. Then we apply the filter (23) in vertical (running from top to bottom) and in horizontal (from left to right) direction.

$$B^{[1]}_{y,x} = X \quad \text{with} \quad X \sim N(0,1) \tag{25}$$

$$B^{[2]}_{y,x} = \sum_{k=0}^{\lfloor 4 \cdot d_\lambda / d_{\mathrm{px}} \rfloor} a_k \cdot B_{y-k,x} \tag{26}$$

$$B^{[3]}_{y,x} = \sum_{k=0}^{\lfloor 4 \cdot d_\lambda / d_{\mathrm{px}} \rfloor} a_k \cdot B_{y,x-k} \tag{27}$$

Next, we apply the second filter (24) on the diagonals of the map. First, we go from the top left to the bottom right and then we go from the bottom left to the top right.

$$B^{[4]}_{y,x} = \sum_{k=0}^{\lfloor 4 \cdot d_\lambda / d_{\mathrm{px}} \rfloor} b_k \cdot B_{y-k,x-k} \tag{28}$$

$$B^{[5]}_{y,x} = \sum_{k=0}^{\lfloor 4 \cdot d_\lambda / d_{\mathrm{px}} \rfloor} b_k \cdot B_{y+k,x-k} \tag{29}$$

After the autocorrelations are applied, the extension space is removed and values of the remaining map are scaled to have the desired distribution μ, σ. The same procedure is repeated for all seven LSPs. However, the decorrelation distance d_λ as well as μ, σ for each parameter can be different. We account for the cross-correlation of the parameters by assembling a matrix $\mathbf{X}$ that contains all cross-correlation values and multiplying its matrix-square-root with the values of the seven parameter maps $\mathbf{B}_{DS} \ldots \mathbf{B}_{EsD}$. The cross-correlation matrix needs to be positive definite to get a unique, real numbered solution.

$$\begin{pmatrix} B_{y,x,DS} \\ \vdots \\ B_{y,x,EsD} \end{pmatrix} = \mathbf{X}^{\frac{1}{2}} \begin{pmatrix} B^{[5]}_{y,x,DS} \\ \vdots \\ B^{[5]}_{y,x,EsD} \end{pmatrix} \tag{30}$$

Last, the users are placed on the maps and the corresponding values for the LSPs are obtained. In this way, we initialize the second part of the channel model, the generator of the individual channel coefficients, with correlated input values for each user. An example output of the correlation map procedure is shown in Fig. 10. A set of 200 mobile terminals is randomly placed in an urban cellular scenario (blue dots). The transmitter is situated in the scenario center (red cross). The map for the DS is displayed as a background image. The initial values of the LSPs for each user position (link level) serve as an input for calculating the channel coefficients. They are simply generated by reading the map-values around the position of the mobile terminal and interpolating between adjacent pixels.

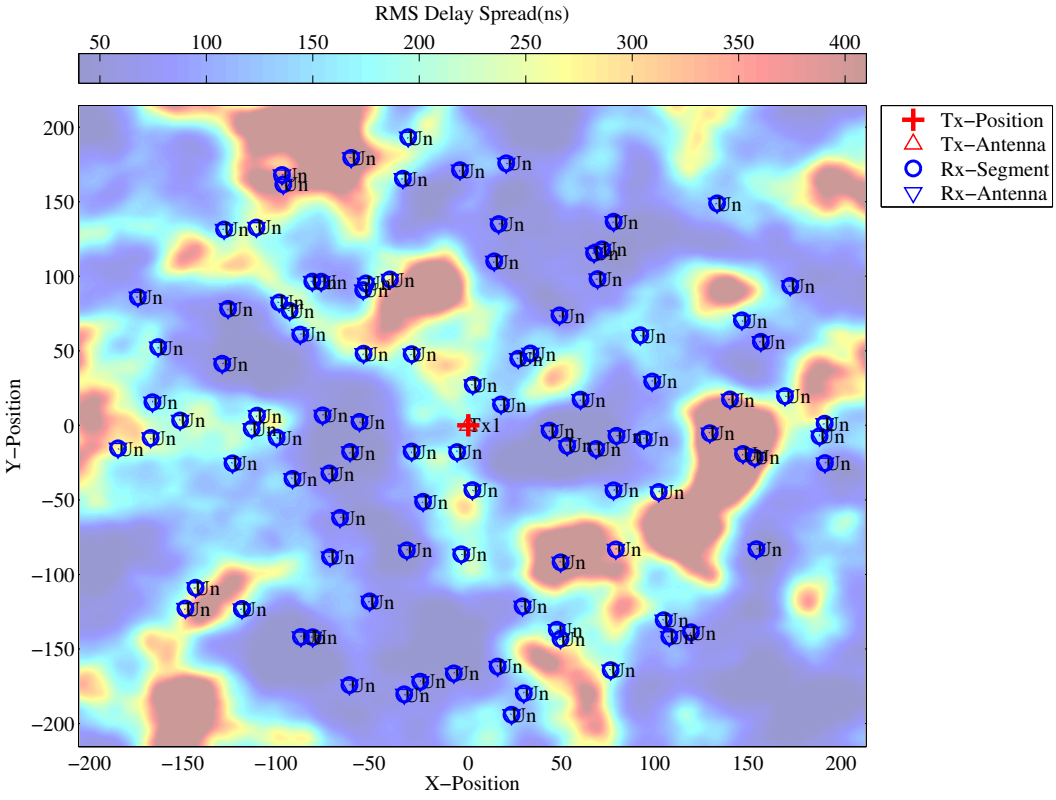

Figure 10: Illustration of the map based generation of the DS

3.2 Calculation of Channel Coefficients

In the model software, the channel coefficients are generated by the *channel builder*. It takes the correlated large scale parameter values as input and calculates the CIR for each user position.

Each scattering cluster is represented by a propagation path which is modeled as a Dirac function in delay domain. A path is made of up of 20 spatially separated *subpaths* according to the sum-of-sinusoids method [PB01]. Path powers, path delays, and angular properties for both sides of the link are modeled as random variables defined through probability density functions (PDFs) and cross-correlations. All parameters, except the fast-fading, are drawn independently in time, in what is termed *drops* (see [3GP05]). Even though a path consists of multiple subpaths in an angular domain, it remains a single tap in delay domain.

3.2.1 Initial Delays and Cluster Powers

Implemented in ⇒ `channel_builder.generate_initial_paths`

Initial delays are drawn randomly from a scenario-dependent delay distribution as

$$\tau_l^{[1]} = -r_\tau \sigma_\tau \ln(X_l) \tag{31}$$

where $X_l \sim \mathrm{uni}(0,1)$ is a uniformly distributed random variable having values in between 0 and 1, σ_τ is the initial DS from the map and r_τ is a proportionality factor. r_τ has been introduced in [3GP11] because σ_τ is influenced by both, the delays τ_l and the powers P_l. r_τ is usually calculated from measurement data. Next, the delays are normalized such that the first delay is zero and then they are sorted in descending order

$$\tau_l^{[2]} = \mathrm{sort}\left\{ \tau_l^{[1]} - \min\left(\tau_l^{[1]}\right) \right\} \tag{32}$$

The NLOS cluster powers are drawn from a single slope exponential power delay profile (PDP) depending on the DS σ_τ and and a random component $Z_l \sim \mathcal{N}(0, \zeta)$. ζ is a scenario-dependent coefficient emulating an additional shadowing process among the clusters in one segment. It is normally obtained from measurements.

$$P_l^{[1]} = \exp\left(-\tau_l \frac{r_\tau - 1}{r_\tau \sigma_\tau} \right) \cdot 10^{\frac{-Z_l}{10}} \tag{33}$$

The power of the first cluster is further scaled according to the initial KF from the map and cluster powers are normalized so that their sum power is unity.

$$P_1^{[1]} = K \cdot \sum_{l=2}^{L} P_l^{[1]} \qquad P_l = P_l^{[1]} / \sum_{l=1}^{L} P_l^{[1]} \tag{34}$$

In the last step, we correct the influence of the KF on the DS which has changed due to the scaling. The DS after applying (34) is calculated by

$$\bar{\sigma_\tau} = \sqrt{ \sum_{l=1}^{L} P_l \cdot (\tau_l)^2 - \left(\sum_{l=1}^{L} P_l \cdot \tau_l \right)^2 } \tag{35}$$

The cluster delays are then obtained by

$$\tau_l = \frac{\sigma_\tau}{\bar{\sigma_\tau}} \cdot \tau_l^{[2]} \tag{36}$$

with σ_τ being the initial DS from the map.

3.2.2 Departure and Arrival Angles

Implemented in ⇒ `channel_builder.generate_initial_angles`
and ⇒ `channel_builder.correction_function`

We calculate four angles for each cluster: the azimuth of departure (AoD, ϕ^d), the elevation of departure (EoD, θ^d), the azimuth of arrival (AoA, ϕ^a) and the elevation of arrival (EoA, θ^a). They share the same calculation method but have a different angular spread σ_ϕ. We assume that the power angular spectrum of all clusters follows a wrapped Gaussian distribution [KMH+07, PMF97].

$$P(\phi) = \frac{1}{\sigma_\phi \sqrt{2\pi}} \exp\left(\frac{-\phi^2}{2\sigma_\phi^2}\right) \tag{37}$$

The wrapping is applied later by (40) when the discrete cluster angles are drawn from the statistics. Since the above formula assumes a continuous spectrum and the channel model, on the other hand, uses discrete paths, we need to correct the variance by a function $C_\phi(L, K)$ depending on the number of clusters L and the KF K. This formula is derived in the Appendix. It ensures that the input variance σ_ϕ is correctly reflected in the generated angles.

We obtain the angles ϕ_l by first normalizing the power angular spectrum so that its maximum has unit power. We can thus omit the scaling factor $1/(\sigma_\phi\sqrt{2\pi})$. We also normalize the path powers P_l (34) so that the strongest peak has unit power which corresponds to an angle $\phi = 0$. All other paths get relative departure- or arrival angles depending on their power

$$\phi_l^{[1]} = \frac{\sigma_\phi}{C_\phi(L, K)} \cdot \sqrt{-2 \cdot \ln(P_l / \max(P_l))} \tag{38}$$

The value σ_ϕ is measured in radians here. Next, we create two random variables X_l and Y_l where $X_l \sim \{-1, 1\}$ is the positive or negative sign and $Y_l \sim \mathcal{N}(0, \sigma_\phi/10)$ introduces a random variation on the angle. Then we calculate

$$\phi_l^{[2]} = X_l \cdot \phi_l^{[1]} + Y_l \tag{39}$$

If the power P_l of a path is small compared to the strongest peak, its angle ϕ_l^b might exceed $\pm\pi$. In this case, we wrap it around the unit circle by a modulo operation

$$\phi_l^{[3]} = \left(\phi_l^{[2]} + \pi \mod 2\pi\right) - \pi \tag{40}$$

In case of elevation spreads, the possible range of elevation angles goes from $-\pi/2$ to $\pi/2$. In this case, we have to correct values of ϕ_l^c outside of this range.

$$\phi_l^{[4]} = \begin{cases} \phi_l^{[3]}, & \text{for el. } \left|\phi_l^{[3]}\right| < \frac{\pi}{2} \text{ and all az. angles;} \\ \pi - \phi_l^{[3]}, & \text{for elevation } \phi_l^{[3]} > \frac{\pi}{2}; \\ \phi_l^{[3]} - \pi, & \text{for elevation } \phi_l^{[3]} < -\frac{\pi}{2}. \end{cases} \tag{41}$$

The positions of the transmitter (Tx) and receiver (Rx) are deterministic and so are the angles of the LOS component. We correct the values of the angles to incorporate this position.

$$\phi_l^{[5]} = \phi_l^{[4]} - \phi_1^{[4]} + \phi^{LOS} \tag{42}$$

Finally, the cluster-path is split into 20 sub-paths to emulate intra cluster angular spreads.

$$\phi_{l,m} = \phi_l^{[5]} + c_\phi \cdot \hat{\phi}_m \tag{43}$$

m is the sub-path index, c_ϕ is the scenario-dependent cluster-wise RMS angular spread and $\hat{\phi}$ is the offset angle of the m^{th} sub-path from Table 11. Furthermore, each of the 20 angle pairs $(\phi_{l,m}^d, \theta_{l,m}^d)$ at the Tx gets coupled with a random angle pair $(\phi_{l,m}^a, \theta_{l,m}^a)$ at the Rx (see [KMH+07]).

Table 11: Offset Angle of the m^{th} Sub-Path from [KMH+07]

Sub-path m	Offset angle $\hat{\phi}_m$ (degrees)	Sub-path m	Offset angle $\hat{\phi}_m$ (degrees)
1,2	± 0.0447	11,12	± 0.6797
3,4	± 0.1413	13,14	± 0.8844
5,6	± 0.2492	15,16	± 1.1481
7,8	± 0.3715	17,18	± 1.5195
9,10	± 0.5129	19,20	± 2.1551

Derivation of the Correction Function The correction function $C_\phi(L, K)$ takes the influence of the K-Factor and the varying number of clusters into account. To approximate the function, we generate the angles as described above. Then we calculate the angular spread from the simulated data and compare the output of the procedure with the given value of σ_ϕ. The angular spread $\tilde{\sigma}_\phi$ is calculated from the given P_l and ϕ_l as [Rap02]

$$\begin{aligned} \tilde{\sigma}_\phi &= 2\pi \cdot \sqrt{1 - \frac{|F_1|^2}{F_0^2}} \\ F_n &= \sum_{l=1}^{L} P_l \cdot \exp(-j \cdot \phi_l \cdot n) \end{aligned} \tag{44}$$

where ϕ_l is the angle calculated by (40) with the correction function set to $C = 1$. F_n is the n-th complex Fourier coefficient. The correction function now follows from comparing $\tilde{\sigma}_\phi$ with σ_ϕ. However, two aspects need to be considered here:

1. Due to the randomization of the angles in (39), we have to take the average angle over a sufficiently large quantity ($\approx$ 1000 realizations) of $\tilde{\sigma}_\phi$. This values is denoted as $\bar{\sigma}_\phi$.

2. Due to the logarithm in (38) and the modulo operation in (40), there is a nonlinear dependency of the angular spread that can be found in the output data and the value given to the model. However, for small values, the relationship can be approximated by a linear function. We define this maximum angular spread $\sigma_\phi^{\max}$ of the linear approximation as the point where the error between the corrected value $\frac{\sigma_\phi}{C_\phi(L,K)}$ and $\bar{\sigma}_\phi$ is 10°.

For each $L \in [2, 42]$ and $K_{[\text{dB}]} \in [-20, 20]$ we now numerically calculate the value of $C_\phi(L, K)$ by

$$C_\phi(L, K) = \frac{1}{\sigma_\phi^{\max}} \cdot \int_0^{\sigma_\phi^{\max}} \frac{\bar{\sigma}_\phi(\sigma_\phi)}{\sigma_\phi} d\sigma_\phi \tag{45}$$

where the σ_ϕ-dependency of $\bar{\sigma}_\phi(\sigma_\phi)$ comes form the individual angles ϕ_l. The function is plotted for different values of L in Fig. 11, left, and tabularized in Tab. 13. Values that are not in the table will be interpolated using linear interpolation.

The plot in Fig. 11, right, compares both, the values given in σ_ϕ and the values calculated by (44) for the final model including the correction function. The parameters for the scatter plot were set to typical values where L ranges from 6 to 30 paths and K ranges from -15 to 15 dB.

The K-Factor has a strong influence on the achievable angular spread. E.g. when the K-Factor is 10, then almost 92% of the energy are focussed in one direction. Thus, it is impossible to get high angular spreads in this case. Tab. 12 gives an overview of the achievable angular spread in azimuth- and elevation direction depending on the K-Factor. The table is valid for realistic path-numbers ranging from 6 to 30 paths.

eMail: quadriga@hhi.fraunhofer.de

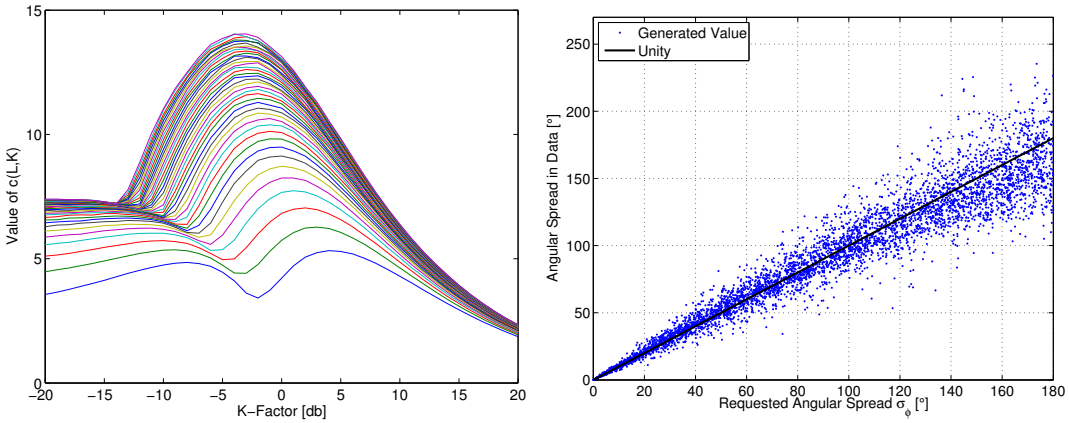

Figure 11: Left: Correction function $C_\phi(L, K)$. Individual lines are for different numbers of paths ranging from 3 (lowest line) to 42 (top line). Right: Comparison of the angular spread σ_ϕ set by the parametrization and the angular spread $\tilde{\sigma}_\phi$ in the data.

Table 12: Maximum Linear Angular Spread vs. K-Factor

K	Az. $\sigma_\phi^{\max}$	El. $\sigma_\phi^{\max}$	K	Az. $\sigma_\phi^{\max}$	El. $\sigma_\phi^{\max}$	K	Az. $\sigma_\phi^{\max}$	El. $\sigma_\phi^{\max}$
-20	206	201	-6	211	209	8	155	151
-18	207	201	-4	209	206	10	143	135
-16	207	202	-2	209	205	12	128	118
-14	205	202	0	200	198	14	113	101
-12	208	203	2	192	190	16	99	84
-10	212	204	4	182	179	18	86	70
-8	213	208	6	170	166	20	74	57

Table 13: Values of $C_\Phi(L, K)$

K/L	3	6	9	12	15	18	21	24	27	30	33	36	39	42
-20	3.59	5.53	6.25	6.62	6.81	6.93	7.04	7.11	7.17	7.13	7.22	7.22	7.25	7.31
-18	3.76	5.63	6.34	6.71	6.78	6.98	7.05	7.07	7.19	7.18	7.23	7.21	7.28	7.26
-16	4.00	5.76	6.36	6.74	6.88	6.96	7.08	7.09	7.18	7.14	7.18	7.24	7.26	7.28
-14	4.24	5.90	6.51	6.78	6.89	6.92	6.99	7.05	7.08	7.16	7.14	7.15	7.06	7.10
-12	4.54	5.98	6.47	6.68	6.74	6.84	6.86	6.92	6.86	7.03	7.38	7.80	8.36	8.77
-10	4.74	5.98	6.38	6.51	6.55	6.66	7.07	7.76	8.54	9.03	9.65	10.12	10.55	10.97
-8	4.83	5.80	5.99	6.27	7.27	8.38	9.12	9.89	10.45	10.90	11.44	11.74	12.10	12.35
-6	4.70	5.30	6.42	7.96	9.13	10.01	10.69	11.36	11.78	12.15	12.60	13.01	13.15	13.47
-4	4.12	5.72	8.00	9.42	10.34	11.09	11.52	12.16	12.57	12.86	13.20	13.39	13.72	14.02
-2	3.42	6.94	8.88	10.01	10.71	11.32	11.90	12.15	12.65	12.95	13.17	13.37	13.53	13.75
0	4.19	7.69	9.07	10.05	10.68	11.09	11.58	11.76	12.12	12.36	12.48	12.84	12.95	13.03
2	5.09	7.66	8.76	9.53	9.98	10.42	10.68	10.99	11.20	11.31	11.45	11.62	11.79	11.87
4	5.33	7.22	8.17	8.67	9.07	9.33	9.53	9.81	9.93	10.01	10.24	10.31	10.44	10.59
6	5.21	6.64	7.26	7.63	7.94	8.10	8.33	8.51	8.62	8.63	8.82	8.89	9.00	9.08
8	4.80	5.85	6.30	6.60	6.83	6.97	7.12	7.23	7.36	7.38	7.43	7.53	7.62	7.67
10	4.29	5.02	5.38	5.59	5.75	5.87	5.97	6.05	6.11	6.21	6.26	6.30	6.33	6.37
12	3.75	4.28	4.54	4.68	4.79	4.90	4.96	5.01	5.10	5.13	5.19	5.22	5.24	5.27
14	3.20	3.59	3.77	3.90	3.97	4.06	4.11	4.17	4.20	4.23	4.26	4.28	4.32	4.35
16	2.70	2.98	3.11	3.21	3.28	3.33	3.37	3.41	3.44	3.46	3.49	3.52	3.54	3.56
18	2.25	2.46	2.56	2.63	2.68	2.72	2.76	2.78	2.80	2.83	2.85	2.86	2.88	2.89
20	1.86	2.01	2.10	2.14	2.18	2.22	2.24	2.26	2.27	2.29	2.31	2.32	2.33	2.34

3.2.3 Drifting of Angles, Delays and Phases

Implemented in ⇒ `channel_builder.get_drifting`

After cluster-delays, powers and angles are known for the initial position, we update their values for each snapshot of the segment. Thus, we get an evolution of the parameters over a short time interval. Since the values of the LSPs and the number of clusters are kept constant, drifting requires that the segment is relatively confined in distance and does not exceed the average autocorrelation distance of the LSPs. A similar concept was already introduced by Baum et. al. in an extension of the SCM [BHS05]. However, it was not incorporated into the WINNER models and no evaluation was reported.

Besides the parameters from steps B and C, drifting requires the position of each antenna element at the Tx and Rx. Additionally, Rx element positions need to be provided for each snapshot separately, depending on the orientation and position of the Rx. The following calculations are then done element-wise where the indices r, t, l, m, s denote the element index of the Rx antenna r and the Tx antenna t, the cluster number l, the sub-path number m and the snapshot number within the current segment s, respectively.

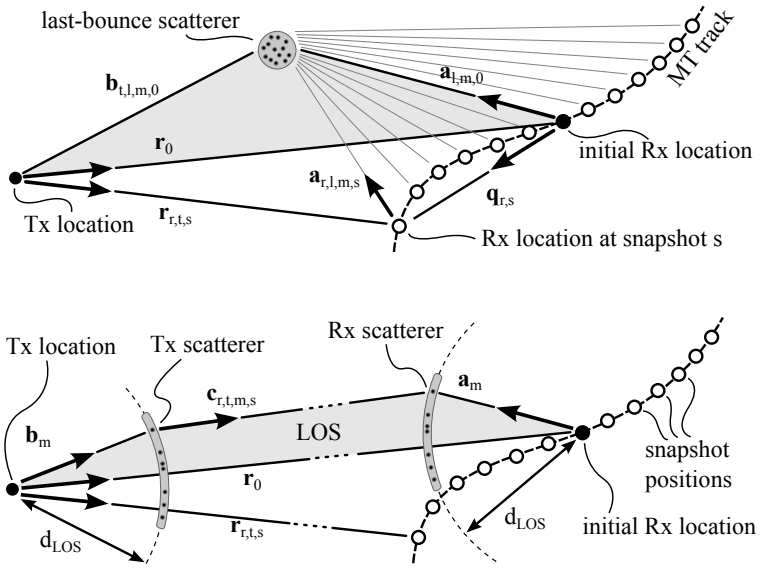

Figure 12: Illustration of the calculation of the scatterer positions and updates of the arrival angles for NLOS (top) and LOS (bottom).

NLOS drifting For the NLOS paths, we calculate the position of the last bounce scatterer (LBS) from the initial arrival angles and the cluster delays. Then we update the angles and path lengths between the last bounce scatterer (LBS) and the terminal for each snapshot on the track. This is done for each antenna element separately. An illustration of the angles and their relations is given in Fig. 12, top.

The first delay is always zero due to (32). Hence, we calculate the total length of the l^{th} path as

$$d_l = \tau_l \cdot c + |\mathbf{r}_0| \tag{46}$$

where $|\mathbf{r}_0|$ is the distance between the Tx and the initial Rx location and c is the speed of light. We assume, that all sub-paths have the same delay and thus the same path length. However, each sub-path has different arrival angles $(\phi^a_{l,m}, \theta^a_{l,m})$. We transform those angles a vector $\hat{\mathbf{a}}_{l,m,0}$ in Cartesian coordinates and obtain

$$\hat{\mathbf{a}}_{l,m,0} = \frac{\mathbf{a}_{l,m,0}}{|\mathbf{a}_{l,m,0}|} = \begin{pmatrix} \cos\phi^a_{l,m} \cdot \cos\theta^a_{l,m} \\ \sin\phi^a_{l,m} \cdot \cos\theta^a_{l,m} \\ \sin\theta^a_{l,m} \end{pmatrix} \tag{47}$$

We approximate the drifting at the Rx by assuming only a single reflection. Hence, Tx, Rx and LBS form a triangle. Since we know d_l, $\mathbf{r}_0$ and $\hat{\mathbf{a}}_{l,m,0}$, we can apply the cosine theorem to calculate the distance $|\mathbf{a}_{l,m,0}|$ between the Rx and LBS[1].

$$\begin{aligned} b_{l,m,0}^2 &= |\mathbf{r}_0|^2 + |\mathbf{a}_{l,m,0}|^2 \\ &\quad -2\,|\mathbf{r}_0|\,|\mathbf{a}_{l,m,0}|\cos\beta_{l,m,0} \\ (d_l - |\mathbf{a}_{l,m,0}|)^2 &= |\mathbf{r}_0|^2 + |\mathbf{a}_{l,m,0}|^2 + 2\,|\mathbf{a}_{l,m,0}|\,\mathbf{r}_0^T\hat{\mathbf{a}}_{l,m,0} \\ |\mathbf{a}_{l,m,0}| &= \frac{d_l^2 - |\mathbf{r}_0|^2}{2\cdot(d_l + \mathbf{r}_0^T\hat{\mathbf{a}}_{l,m,0})} \end{aligned} \tag{48}$$

Now we can calculate the vector $\mathbf{a}_{r,l,m,s}$ for the Rx antenna element r at snapshot s. The element position includes the orientation of the antenna array with respect to the moving direction of the Rx. Hence, the vector $\mathbf{q}_{r,s}$ points from the initial Rx location to the r^{th} antenna element at snapshot s.

$$\mathbf{a}_{r,l,m,s} = \mathbf{a}_{l,m,0} - \mathbf{q}_{r,s} \tag{49}$$

Now, we can obtain an update of the arrival angles by transforming $\mathbf{a}_{r,l,m,s}$ back to spherical coordinates.

$$\phi_{r,l,m,s}^a = \arctan_2\left\{a_{r,l,m,s,y}; a_{r,l,m,s,x}\right\} \tag{50}$$

$$\theta_{r,l,m,s}^a = \arcsin\left\{\frac{a_{r,l,m,s,z}}{|\mathbf{a}_{r,l,m,s}|}\right\} \tag{51}$$

We assume a static scattering environment. Thus, the departure angles at the Tx do not change. We also assume that the distance between the Tx and the first scatterer is large compared to the Tx array dimension. Hence, we use the same departure angles for all Tx elements. The phases and path delays, however, depend on the total path length $d_{r,t,l,m,s}$. To obtain this value, we calculate the vector $\mathbf{b}_{t,l,m,0}$ from the vectors $\mathbf{r}_{r,t,s}$ and $\mathbf{a}_{r,l,m,s}$ at $r = s = 1$.

$$\mathbf{b}_{t,l,m,0} = \mathbf{r}_{1,t,1} + \mathbf{a}_{1,l,m,1} \tag{52}$$

$$d_{r,t,l,m,s} = |\mathbf{b}_{t,l,m,0}| + |\mathbf{a}_{r,l,m,s}| \tag{53}$$

Finally, we calculate the phase ψ and path delays τ.

$$\psi_{r,t,l,m,s} = \frac{2\pi}{\lambda}\cdot(d_{r,t,l,m,s} \bmod \lambda) \tag{54}$$

$$\tau_{r,t,l,s} = \frac{1}{20\cdot c}\sum_{m=1}^{20} d_{r,t,l,m,s} \tag{55}$$

LOS drifting The direct component is handled differently since there are no discrete scatterers. LOS fading, however, can occur if there are objects in the first Fresnel zone of the propagation path. To simplify the calculations, we assume that those objects have a constant distance to the Rx and are only separated by their angular distribution. They are placed on a circle with radius d_{LOS} as depicted in Fig. 12, bottom. We update the departure- and arrival angles based on the vector $\mathbf{r}_{r,t,s}$ which points from the location of the Tx element t to the location of the Rx element r at snapshot s.

$$\phi_{t,1,m,s}^d = \arctan_2\left\{r_{r,t,s,y}, r_{r,t,s,x}\right\} + c_{\text{AoD}}\cdot\hat{\phi}_m \tag{56}$$

$$\theta_{t,1,m,s}^d = \arcsin\left\{r_{r,t,s,z}/\,|\mathbf{r}_{r,t,s}|\right\} + c_{\text{EoD}}\cdot\hat{\phi}_m \tag{57}$$

$$\phi_{r,1,m,s}^a = \arctan_2\left\{-r_{r,t,s,y}, -r_{r,t,s,x}\right\} + c_{\text{AoA}}\cdot\hat{\phi}_m \tag{58}$$

$$\theta_{r,1,m,s}^a = \arcsin\left\{-r_{r,t,s,z}/\,|\mathbf{r}_{r,t,s}|\right\} + c_{\text{EoA}}\cdot\hat{\phi}_m \tag{59}$$

[1] We substitute $\cos\beta_{l,m,0}$ with $-\mathbf{r}_0^T\hat{\mathbf{a}}_{l,m,0}/|\mathbf{r}_0|$ since we are at the Rx position looking towards the Tx.

eMail: quadriga@hhi.fraunhofer.de

The phases and delays, are determined by the length of the vector $\mathbf{c}_{r,t,m,s}$ in Fig. 12, bottom. This vector points from a random scatterer at the Tx to a random scatterer at the Rx. For the sake of simplicity, we define the vectors $\mathbf{a}_m$ and $\mathbf{b}_m$ to be position-independent, i.e. their origin is always relative to the Tx and Rx antenna element. The vector $\mathbf{c}_{r,t,m,s}$ and the path length $d_{r,t,m,s}$ then follow from

$$\mathbf{c}_{r,t,m,s} = -\mathbf{b}_m + \mathbf{r}_{r,t,s} + \mathbf{a}_m \tag{60}$$

$$d_{r,t,1,m,s} = 2 \cdot d_{\mathrm{LOS}} + |\mathbf{c}_{r,t,m,s}| \tag{61}$$

Finally, the LOS phase $\psi_{r,t,1,m,s}$ and the delay $\tau_{r,t,1,s}$ can be calculated using (54) and (55).

3.2.4 Geometric Polarization

Implemented in ⇒ `channel_builder.get_channels` and ⇒ `channel_builder.generate_xpr`

As for the angles and the delays, the polarization is calculated in a geometric way. This is done for each (sub-)path, for each snapshot and for each antenna pair (r, t) separately. For the sake of simplicity, we omit the indices l, m, s for the path, subpath and snapshot in the following. Due to the sheer number of computations, calculating the drifting polarization is also the most computing-intense task. To reduce this complexity, the polarization is only updated when the arrival-angle changed more than 0.2 degree since the last update[2].

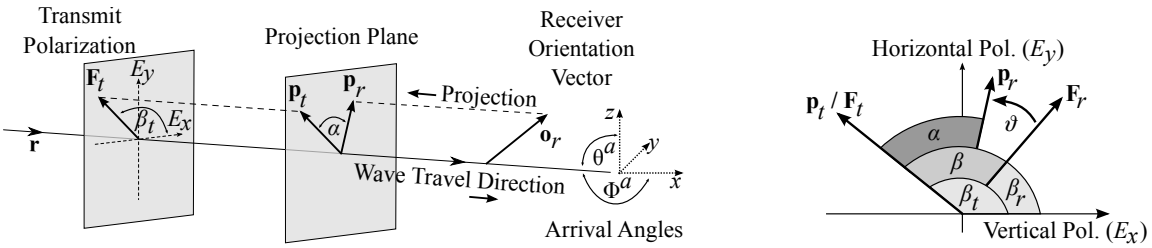

Figure 13: Illustration of the angles and vectors used for the computation of the geometric LOS polarization. Left: Scheme of the projection. Right: Angles on the projection plane.

Model for the LOS component The principle of the model is depicted in Fig. 13. The wave travel direction $\mathbf{r}$ is determined by the AoA at the receiver given in azimuth ϕ^a and elevation θ^a direction. The transmit polarization vector $\mathbf{F}_t$ results from the beam pattern (7) and lies in the plane perpendicular to $\mathbf{r}$. Note that our method is only valid for linearly polarized waves. Circular polarization can be obtained by combining two linear elements with a phase offset. The receive antenna can have any orientation in 3D space. Thus, we need additional information on the element orientation which is realized by a vector $\mathbf{o}_r$. $\mathbf{o}_r$ represents the linear receiver polarization. However, this vector does not lie in the same plane as $\mathbf{F}_t$ does. The polarization vector $\mathbf{p}_r$ results from a projection of the orientation vector $\mathbf{o}_r$ on the plane perpendicular to the travel direction. Due to the orientation mismatch between transmitter and receiver, there will be an offset between $\mathbf{p}_r$ and the vector obtained from the receiver beam pattern $\mathbf{F}_r$. This offset is compensated by introducing a virtual polarizer that turns the polarization state of the wave in a way that it matches the orientation of the receiver. Since we do not want to change the amplitude of the wave (this is handled by other parts of the model) nor the type of polarization, we need to compute a rotation matrix. Thus, we have to determine the rotation angle ϑ. This can be done by the following procedure.

1. To simplify the computations, we rotate the coordinate system such that the wave travel direction $\mathbf{r}$ lies in y-direction (i.e. $\mathbf{r}^+ = (0, 1, 0)^T$). Thus, we need to rotate the orientation vector $\mathbf{o}_r$ by $p = -\phi^a - \pi/2$ in

[2] This value can be changed in `simulation_parameters.drifting_update_threshold`

azimuth direction and $q = \theta^a$ in elevation direction.

$$\mathbf{o}_r^+ = \begin{pmatrix} \cos p & -\sin p & 0 \\ \cos q \cdot \sin p & \cos q \cdot \cos p & -\sin q \\ \sin q \cdot \sin p & \sin q \cdot \cos p & \cos q \end{pmatrix} \cdot \mathbf{o}_r \tag{62}$$

2. We calculate the projection of the receiver orientation vector on the projection plane. Since the projection plane lies now in the x-z-plane due to the rotation of the coordinate system, we simply omit the y component of $\mathbf{o}_r^+$, switch the x and z component and normalize the resulting vector to unit length. The switching is done to obtain the same orientation as in (7).

$$\mathbf{p}_r = \begin{pmatrix} o_{rz}^+ \\ o_{rx}^+ \end{pmatrix} / \left| \begin{pmatrix} o_{rz}^+ \\ o_{rx}^+ \end{pmatrix} \right| \tag{63}$$

3. We get the transmit polarization vector $\mathbf{p}_t$ by normalizing the field pattern vector $\mathbf{F}_t$ (7) to unit length. $\mathbf{F}_t$ is already in the same plane as $\mathbf{p}_r$ due to the coordinate rotation.

$$\mathbf{p}_t = \mathbf{F}_t / \left| \mathbf{F}_t \right| \tag{64}$$

4. We calculate the angle α between the two vectors.

$$\alpha = \arccos \left(\mathbf{p}_r^T \cdot \mathbf{p}_t \right) \tag{65}$$

5. We obtain the angles β_t and β_r from the field patterns of the transmitter and receiver, respectively. β is the angle between the x-axis of the polarization plane and the polarization vector. Note that E_x in Fig. 13 is the vertical and E_y is the horizontal polarized component.

$$\beta_t = \arctan_2 \left\{ F_{tH} \left(\phi^d, \theta^d \right), F_{tV} \left(\phi^d, \theta^d \right) \right\} \tag{66}$$

$$\beta_r = \arctan_2 \left\{ F_{rH} \left(\phi^a, \theta^a \right), F_{rV} \left(\phi^a, \theta^a \right) \right\} \tag{67}$$

6. The difference between the angles β_t and β_r is the polarization mismatch β between the receiver and the transmitter if both antenna elements were aligned on the same optical axis. The angle α, however takes the different orientation of the receive antenna into account. We thus need to rotate the polarization of the receiver by an angle of

$$\vartheta_{r,t} = \beta_t - \beta_r - \alpha \tag{68}$$

7. The channel coefficients are now calculated according to (6) where the polarization coupling matrix $\mathbf{M}$ notes

$$\mathbf{M}(\vartheta_{r,t}) = \begin{pmatrix} \cos \vartheta_{r,t} & \sin \vartheta_{r,t} \\ -\sin \vartheta_{r,t} & \cos \vartheta_{r,t} \end{pmatrix} \tag{69}$$

Model for the NLOS components For the NLOS components, the transmitted signal undergoes some diffraction, reflection or scattering before reaching the receiver. Following the common Fresnel formula in electrodynamics, the polarization direction can be changed at the boundary surface between two dielectric media. T. Svantesson [Sva01] provided a method for modeling the polarization of a reflected wave where the polarization coupling is a function of several geometric parameters such as the orientation of the scatterers. However, these parameters are not generally available in the SCM. In addition to that, only metallic reflections keep the polarization unchanged. Reflections at dielectric media can cause changes of the polarization being a function of the complex-valued dielectric constant of the media and of the angle of incidence. Hence, not only the polarization angle might change, but also the polarization type. In order to address this issue,

studies of the polarizations effects in individual scattering clusters in several outdoor- and indoor scenarios were done [MTIO09, QOHDD10, PHL+11]. The published results indicate that, in many cases, scattering preserves the polarization quiet well. However, since only the powers of the elements in the polarization coupling matrix were analyzed, no conclusions can be drawn on how elliptic the polarization of the scattered wave becomes.

We assume that the polarization coupling matrix $\mathbf{M}$ for the NLOS components can be described by a combination of linear transformations. Hence, we can take advantage of the existing findings of the XPR. If the XPR is identical for both polarization directions (such as in the WINNER parameter tables), then we can follow the approach from Zhou et. al. [ZRP+05] and calculate an additional NLOS rotation matrix $\mathbf{M}_\gamma$ as

$$\mathbf{M}_\gamma = \begin{pmatrix} m_{vv} & m_{vh} \\ m_{hv} & m_{hh} \end{pmatrix} = \begin{pmatrix} \cos\gamma & -\sin\gamma \\ \sin\gamma & \cos\gamma \end{pmatrix} \tag{70}$$

Following the notations in [OCGD08], we get

$$\mathrm{XPR} = \frac{|m_{vv}|^2}{|m_{hv}|^2} = \frac{|m_{hh}|^2}{|m_{vh}|^2} = \frac{(\cos\gamma)^2}{(\sin\gamma)^2} = (\cot\gamma)^2 \tag{71}$$

$$\gamma = \operatorname{arccot}\left(\sqrt{\mathrm{XPR}}\right) \tag{72}$$

However, when the XPR is different for the vertical and horizontal component [OCGD08, QOHDD10], then we get three parameters:

$$\mathrm{XPR}_v = \frac{|m_{vv}|^2}{|m_{hv}|^2} \quad \mathrm{XPR}_h = \frac{|m_{hh}|^2}{|m_{vh}|^2} \quad \mathrm{CPR} = \frac{|m_{vv}|^2}{|m_{hh}|^2}$$

In order to fulfill all three, we can combine two rotations, one for the vertical and one for the horizontal component, with a scaling operation. We convert XPR_v and XPR_h to rotation angles γ_v and γ_h using (72) and calculate $\mathbf{M}_\gamma$ to

$$\mathbf{M}_\gamma = \begin{pmatrix} \cos\gamma_v & -\tan\gamma_h \cdot \cos\gamma_v \cdot \frac{1}{\sqrt{\mathrm{CPR}}} \\ \sin\gamma_v & \cos\gamma_v \cdot \frac{1}{\sqrt{\mathrm{CPR}}} \end{pmatrix} \tag{73}$$

Elliptic polarization is obtained, when there is a phase difference κ between the horizontal and the vertical component. This is included by a scaling matrix

$$\mathbf{M}_\kappa = \begin{pmatrix} 1 & 0 \\ 0 & \exp j\kappa \end{pmatrix} \tag{74}$$

The antenna-dependent parameters β_r, β_t and α are handled as in the LOS case using (68) which results in a rotation matrix $\tilde{\mathbf{M}}(\vartheta)$. The transformations are then combined to

$$\mathbf{M} = \frac{\sqrt{2}}{||\mathbf{M}_\gamma||_F} \cdot \tilde{\mathbf{M}}(\vartheta) \cdot \mathbf{M}_\gamma \cdot \mathbf{M}_\kappa \tag{75}$$

The normalization with the Frobenius norm $||\mathbf{M}_\gamma||_F$ ensures that $\mathbf{M}$ does not change the power of the multipath component.

eMail: quadriga@hhi.fraunhofer.de

3.2.5 Calculation of the Channel Coefficients

Implemented in ⇒ `channel_builder.get_channels` and ⇒ `array.interpolate`

Next, we combine antenna patterns, polarization and phases to calculate initial channel coefficients for each snapshot of a segment. The antennas are defined by their polarimetric response **F** containing vertical and the horizontal polarization in spherical coordinates [NKS+07].

$$\mathbf{F}(\phi,\theta) = \begin{pmatrix} F_V(\phi,\theta) \\ F_H(\phi,\theta) \end{pmatrix} \tag{76}$$

Since we already know the arrival- and departure angles of each MPC, we can combine the response from both, the Tx and Rx antenna with the polarization rotation and get the coefficient

$$g^{[\mathrm{pol}]}_{r,t,l,m,s} = \mathbf{F}_r(\phi^a,\theta^a)^T \cdot \mathbf{M} \cdot \mathbf{F}_t(\phi^d,\theta^d) \tag{77}$$

Each MPC has a random initial phase ψ^0. Hence, by summing up the 20 sub-paths in order to get one path per cluster, we get a random cluster power. This is compensated by normalization where we first sum up the complex phases and then average the power over all S snapshots of the segment. We calculate the "raw" channel coefficients as

$$\psi^+_{r,t,l,m,s} = \exp\left(-j\psi^0_{l,m} - j\psi_{r,t,l,m,s}\right) \tag{78}$$

$$P^{[\mathrm{norm}]}_{r,t,l} = \frac{1}{S}\sum_{s=1}^{S}\left(\sum_{m=1}^{20}\psi^+_{r,t,l,m,s}\right)^2 \tag{79}$$

$$g^{[\mathrm{raw}]}_{r,t,l,s} = \sqrt{\frac{P_l}{P^{[\mathrm{norm}]}_{r,t,l}}}\cdot\sum_{m=1}^{20} g^{[\mathrm{pol}]}_{r,t,l,m,s}\cdot\psi^+_{r,t,l,m,s} \tag{80}$$

where P_l is the initial power assigned to each cluster.

3.2.6 Path Gain, Shadow Fading and K-Factor

Implemented in ⇒ `channel_builder.get_channels` and ⇒ `parameter_set.get_sf_profile`

Now, we apply the path gain (PG), the shadow fading (SF) and the KF. A common PG model for macro-cellular settings is given by M. Hata [Hat80] where the PG scales with the logarithm of the distance d (in units of meters) between BS and terminal.

$$\mathrm{PG}_{[\mathrm{dB}]} = -A\cdot 10\log_{10} d_{[\mathrm{m}]} - B \tag{81}$$

A and B are scenario-specific coefficients which are typically determined by measurements. A often varies between 2 and 4, depending on the propagation conditions, the BS height and other factors.

The values for the SF and the KF are obtained from the map by an interpolation of the surrounding pixels at the position of the s^{th} snapshot. The KF at the initial position is already included due to the scaling in (34). Thus, we have to take this into account and scale the power accordingly.

$$g_{r,t,l,s} = P^{[\mathrm{MT}]}_s \cdot \begin{cases} \sqrt{\frac{K_s}{K_0}}\cdot g^{[\mathrm{raw}]}_{r,t,1,s} & \text{for } l=1; \\ g^{[\mathrm{raw}]}_{r,t,l,s} & \text{otherwise.} \end{cases} \tag{82}$$

$$P^{[\mathrm{MT}]}_s = \sqrt{10^{0.1\cdot\mathrm{PG}_{[\mathrm{dB}]_s}+0.1\cdot\mathrm{SF}_{[\mathrm{dB}]_s}}}\cdot\sqrt{1+P_1\left(\frac{K_s}{K_0}-1\right)}$$

K_s and $\mathrm{SF}_{[\mathrm{dB}]_s}$ are the interpolated values for the KF and the SF from the map, K_0 is the KF at the initial position, $\mathrm{PG}_{[\mathrm{dB}]_s}$ is the path gain at the MT position (81) and P_1 is the power of the LOS cluster from (34).

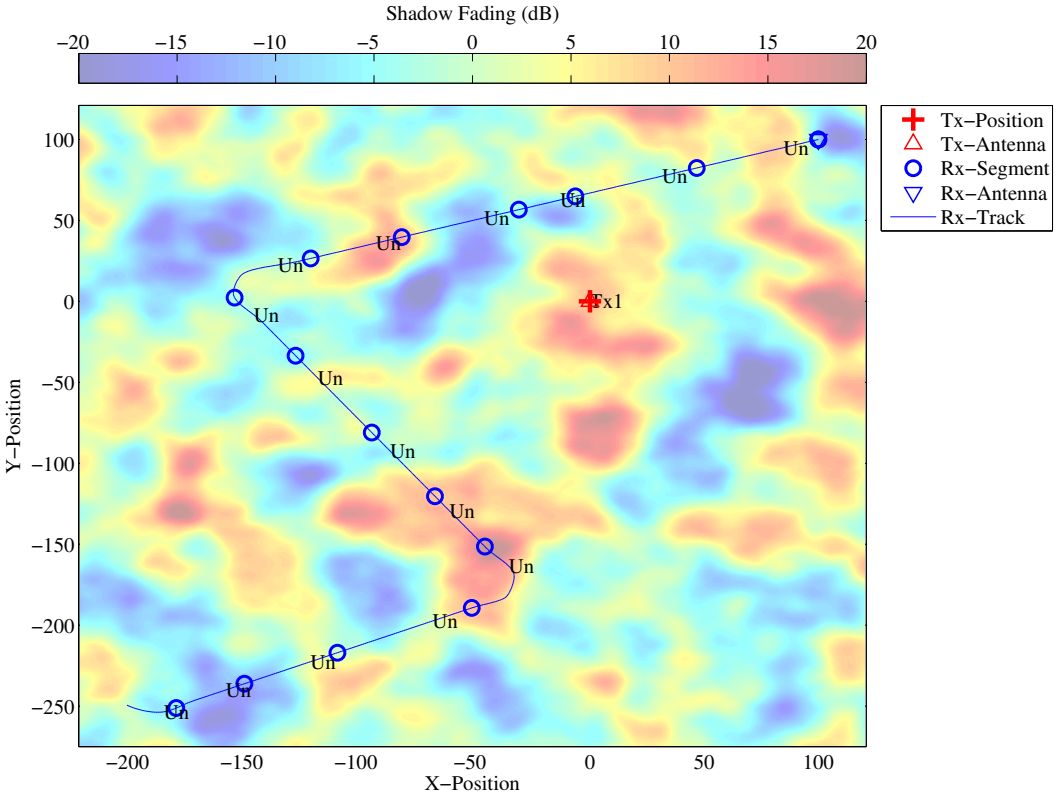

Figure 14: Illustration of the SF drifting along a terminal track

3.2.7 Transitions between Segments

Until now, the calculations were done for each segment of a MT trajectory independently. Longer sequences are created by merging the channels from adjacent segments into one long sequence. The basic idea comes from the documentation of the WINNER II model [KMH+07]. However, it was neither implemented nor tested. Our implementation requires overlapping segments as depicted in Fig. 15, top. Like in a movie, the transition is carried out by ramping down the power of paths in the old segment and at the same time ramping up the power of paths from the new segment. Hence, this process describes the birth and death of clusters along the trajectory. In order to keep the computational overhead low, we split the overlapping part into several sub-intervals equal to the minimum number of clusters in both segments. During each sub-interval, the power of one old cluster ramps down and one new cluster ramps up. We model power ramps by a squared sine function to allow smooth transitions.

$$w^{[\mathrm{sin}]} = \sin^2\left(\frac{\pi}{2} \cdot w^{[\mathrm{lin}]}\right) \tag{83}$$

$w^{[\mathrm{lin}]}$ is a linear ramping function ranging from 0 to 1. $w^{[\mathrm{sin}]}$ is the corresponding sine-shaped ramping function having the advantage of a constant slope at the points 0 and 1, which prevents inconsistencies at the edges of the intervals. If the number of clusters is different in both segments, clusters are ramped up or down without a counterpart from the new/old segment. The ramp is then stretched over the whole overlapping area. For the LOS path, we continuously adjust power and phase over the overlapping area since it has the same delay in both segments.

In order to minimize the impact of the transition on the instantaneous values of the LSPs, paths need to be carefully matched. For example, if a path with a small delay ramps down and a similarly strong path with a longer delay ramps up, then the DS increases. This increase (or decrease) can fluctuate randomly along the merging interval. To balance it out, we pair paths from both segments that minimize the fluctuations. This is done by first determining the values of the DS before and after the transition. Then, we calculate a

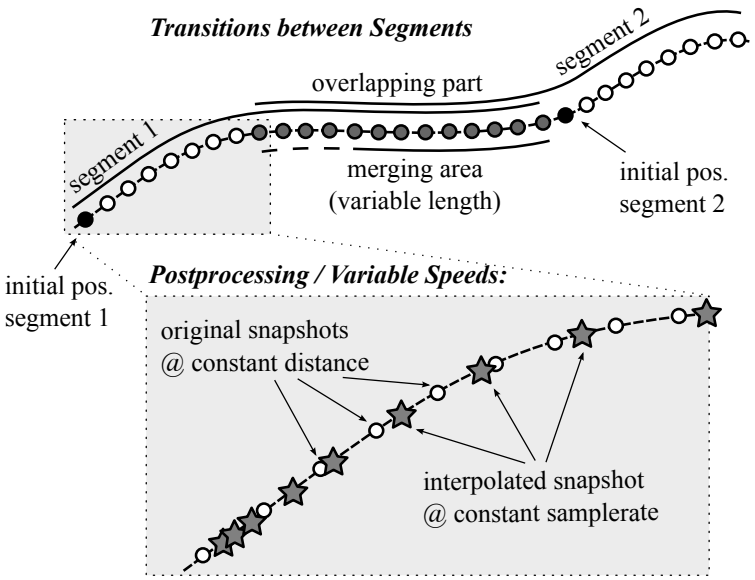

Figure 15: Top: Illustration of the overlapping area used for calculating the transitions between segments (step G); Bottom: Illustration of the interpolation to to obtain variable MT speeds (step H)

target DS for each sub-interval. For example, if the old segment yields a DS of 200 ns and the new segment has 400 ns, then the target-DS will be 220 ns for the first sub-interval, 240 ns for the second and so on. Then we look for a combination of paths to ramp up/down in each sub-interval that best matches the DS along the overlapping area to the target area in a mean square sense.

3.2.8 Postprocessing / Variable Speeds

Realistic channel traces incorporate arbitrary speeds, accelerations and decelerations. Provided that the channel sampling theorem is fulfilled, we can interpolate the coefficients as it is illustrated in Fig. 15, bottom. The white dots represent the snapshots at a constant distance. However, the sample points (gray stars) can have an unequal spacing, e.g. for an accelerated movement. Each sample point in the time domain (given in units of seconds) has a corresponding sample point on the track (in units of meters). The amplitudes and phases of the channel coefficients are interpolated separately using a cubic spline interpolation. The path delays are interpolated with a piecewise cubic hermite interpolating polynomial.

eMail: quadriga@hhi.fraunhofer.de

A Tutorials

In the following, we provide a variety of tutorials that can get you started with QuaDRiGa. You can also use the MATLAB Help to access these files.

A.1 Network Setup and Parameter Generation

The channel model class 'parameter_set' generates correlated values for the LSPs. The channel builder then uses those values to create coefficients that have the specific properties defined in 'parameter_set'. One important question is therefore: Can the same properties which are defined in 'parameter_set' also be found in the generated coefficients? This is an important test to verify, if all components of the channel builder work correctly.

Channel model setup and coefficient generation We first set up the basic parameters. We do not need drifting here, since no time varying channels are generated.

```
close all
clear all

set(0,'defaultTextFontSize', 14)
set(0,'defaultAxesFontSize', 14)

s = simulation_parameters;
s.center_frequency = 2.53e9;
s.sample_density = 2;
s.use_absolute_delays = 1;
s.drifting_precision = 0;
```

We have one transmitter and 250 receiver positions. Each receiver gets a specific channel. However, the receivers LSPs will be correlated. We use omni directional antennas at all terminals.

```
l = layout( s );
l.no_rx = 250;
l.randomize_rx_positions( 200 , 1.5 , 1.5 , 1 );  % 200 m radius, 1.5 m Rx height
l.track.set_scenario('BERLIN_UMa_NLOS');

l.tx_position(3) = 25;                            % 25 m tx height
l.tx_array.generate( 'omni' );
l.rx_array = l.tx_array;

l.visualize([],[],0);
view(-33, 60);
```

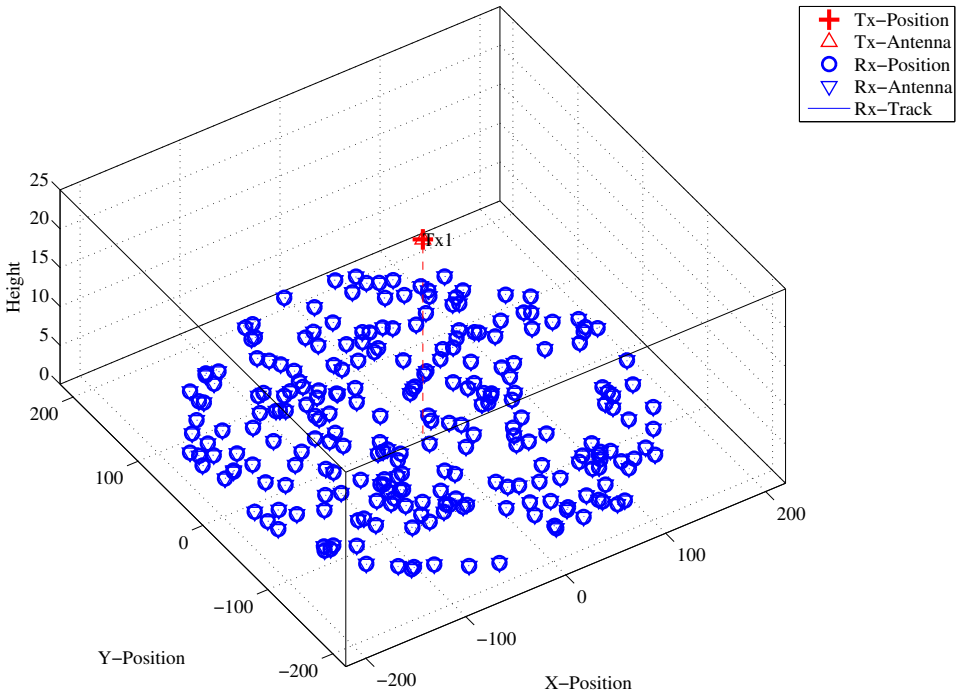

Figure 16: Distribution of the users in the scenario.

We set up the scenario such that there is no XPR. I.e. all vertical polarized paths will remain vertical after a reflection. The same result would be achieved with a perfectly X-polarized array at the receiver and summing up the power over all elements. We further increase the KF to have a wider spread. This allows us to study the parameters at a wider range when evaluating the results.

```
p = l.create_parameter_sets(0);
p.plpar = [];
p.scenpar.xpr_mu    = 100;              % Disable XPR
p.scenpar.xpr_sigma = 0;
p.scenpar.KF_mu     = 5;                % Increase KF-Range
p.scenpar.KF_sigma  = 15;
p.scenpar.DS_mu     = log10(0.6e-6);    % Median DS = 600 ns
p.scenpar.DS_sigma  = 0.3;              % 300-1200 ns range
p.update_parameters;

c = p.get_channels;

coeff = squeeze( cat( 1, c.coeff ) );
delay = permute( cat(3,c.delay) , [3,1,2] );
```

```
Parameters    [ooooooooooooooooooooooooooooooooooooooooooooooooooo]     5 seconds
Channels      [ooooooooooooooooooooooooooooooooooooooooooooooooooo]     8 seconds
```

Results and discussion In the following four plots, we extract parameters from the generated coefficients and compare them with the initial ones which were generated by the 'parameter_set' object (P). The values in (P) can be seen as a request to the channel builder and the values in the generated coefficients (C) as a delivery.

We first calculate the SF from the channel data by summing up the power over all 20 taps. We see, that the values are almost identical.

```
sf = sum(mean( abs(coeff).^2 ,3),2);

figure
plot(-35:35,-35:35,'k')
hold on
plot([-35:35]+3,-35:35,'--k')
plot([-35:35]-3,-35:35,'--k')
plot( 10*log10(p.sf') , 10*log10(sf) , '.'  )
hold off
axis([ -25 , 25 , -25, 25 ])

legend('Equal','+/- 3dB',4)
xlabel('SF_P [dB]');
ylabel('SF_C [dB]');
title('Shadow Fading - Requested vs. generated value');
```

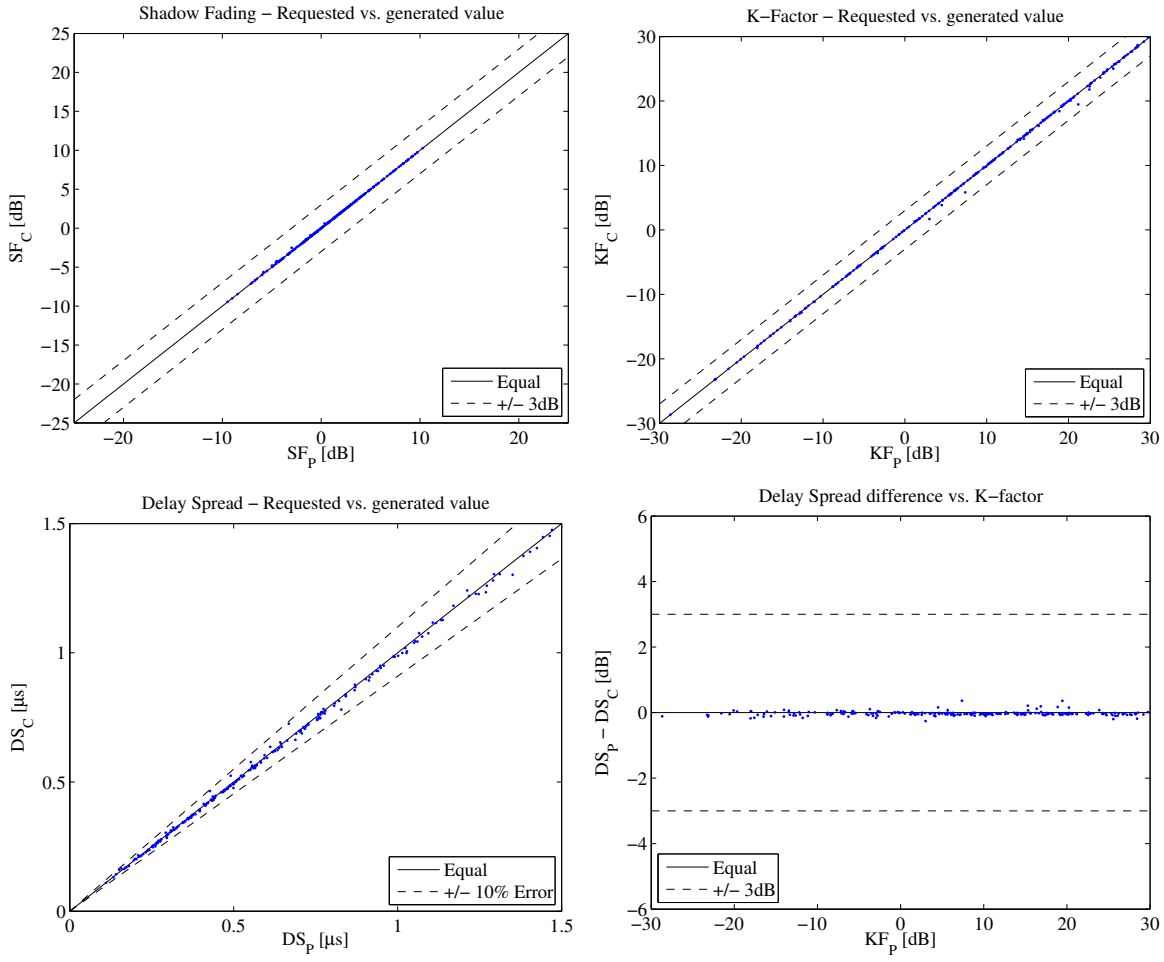

Figure 17: Comparison of input values and simulation results

Next, we repeat the same calculation for the K-Factor. Again, we see that the values are almost identical.

```
p_nlos = sum(mean( abs(coeff(:,2:end,:)).^2 ,3),2);
p_los  = mean( abs(coeff(:,1,:)).^2 ,3);
kf = p_los./p_nlos;

figure
plot(-35:35,-35:35,'k')
hold on
plot([-35:35]+3,-35:35,'--k')
plot([-35:35]-3,-35:35,'--k')
plot( 10*log10(p.kf') , 10*log10(kf) , '.'  )
hold off
axis([ -30 , 30 , -30, 30 ])

legend('Equal','+/- 3dB',4)
xlabel('KF_P [dB]');
ylabel('KF_C [dB]');
title('K-Factor - Requested vs. generated value');
```

Now we repeat the calculation for the RMS delays spread.

```
pow_tap = abs(coeff).^2;
pow_sum = sum(pow_tap,2);
mean_delay = sum( pow_tap.*delay,2) ./ pow_sum;
ds = sqrt( sum( pow_tap.*delay.^2 ,2)./ pow_sum - mean_delay.^2 );
ds = mean(ds,3);

figure
plot([0:0.1:2],[0:0.1:2],'k')
hold on
plot([0:0.1:2]*1.1,[0:0.1:2],'--k')
plot([0:0.1:2],[0:0.1:2]*1.1,'--k')
plot( p.ds'*1e6 , (ds')*1e6 , '.'  )
hold off
axis([ 0,1.5,0,1.5 ])

legend('Equal','+/- 10% Error',4)
xlabel('DS_P [\mus]');
ylabel('DS_C [\mus]');
title('Delay Spread - Requested vs. generated value');
```

The following plot shows the RMSDS of the requested and generated values (in dB) vs. the K-factor. A value of +3 means, that the RMSDS of the generated coefficients is twice a high as in the parameter_set (P). We see, that for a K-Factor of up to 30 dB, the DS difference is small (less than 3 dB).

```
figure
plot([-35,35],[0,0],'k')
hold on
plot([-35,35],[-3,-3],'--k')
plot([-35,35],[3,3],'--k')
plot( 10*log10(p.kf'), 10*log10(ds./p.ds') , '.' )
hold off
axis([ -30 , 30 , -6 6 ])

legend('Equal','+/- 3dB',3)
xlabel('KF_P [dB]');
ylabel('DS_P - DS_C [dB]');
title('Delay Spread difference vs. K-factor');
```

```
close all
disp(['QuaDRiGa Version: ',simulation_parameters.version])
```

```
QuaDRiGa Version: 1.0.1-145
```

A.2 Simulating a Measured Scenario

This script recreates a measured drive test from the Park Inn Hotel at Berlin Alexanderplatz. The transmitter was at the rooftop of the hotel while the mobile receiver was moving south on Grunerstraße. A simplified version of the scenario is recreated in the simulation where the scenarios along the track were classified by hand.

Channel model setup and coefficient generation First, we set up the channel model.

```
set(0,'defaultTextFontSize', 14)
set(0,'defaultAxesFontSize', 14)
RandStream.setGlobalStream(RandStream('mt19937ar','seed',1));

close all
clear all

s = simulation_parameters;                          % Basic simulation parameters
s.center_frequency = 2.185e9;
s.sample_density = 2;
s.use_absolute_delays = 1;

t = track('linear',500,-135*pi/180);                % Track with 500m length, direction SE
t.initial_position = [120;-120;0];                  % Start position
t.interpolate_positions( 1 );                       % Interpolate to 1 sample per meter

t.segment_index = [1    45   97   108  110  160  190  215  235  245 ...
    280  295  304  330  400  430 ];                  % Set segments (states)

Sl = 'MIMOSA_10-45_LOS';
Sn = 'MIMOSA_10-45_NLOS';
t.scenario      = {Sn,Sl,Sn,Sl,Sn,Sn,Sn,Sl,Sn,Sl,Sn,Sl,Sn,Sn,Sn,Sn};
t.interpolate_positions( 3 );

l = layout( s );
l.tx_position = [0;0;125];                          % Set the position of the Tx
l.track = t;                                        % Set the rx-track

l.tx_array = array('rhcp-lhcp-dipole');             % Generate Tx antenna
l.tx_array.rotate_pattern(30,'y');                  % 30 deg Tilt
l.tx_array.rotate_pattern(-90,'z');                 % point southwards

l.rx_array = array('rhcp-lhcp-dipole');             % Rx-Antenna
l.rx_array.rotate_pattern(-90,'y');                 % point skywards

l.visualize;
view(-33, 45);

lnk = [ l.tx_position, ...
    l.track.positions(:,l.track.segment_index(2))+l.track.initial_position ];

hold on
plot3( lnk(1,:),lnk(2,:),lnk(3,:) , '--' )
hold off
```

Generate channel coefficients Next, we calculate the channel coefficients.

```
[p,cb] = l.create_parameter_sets(0);
p(2).scenpar.NumClusters = 14;
p.update_parameters;

c = cb.get_channels;
cn = c.merge(0.2);
```

```
Parameters   [ooooooooooooooooooooooooooooooooooooooooooooooooooo]    31 seconds
Channels     [ooooooooooooooooooooooooooooooooooooooooooooooooooo]    15 seconds
Merging      [ooooooooooooooooooooooooooooooooooooooooooooooooooo]     2 seconds
```

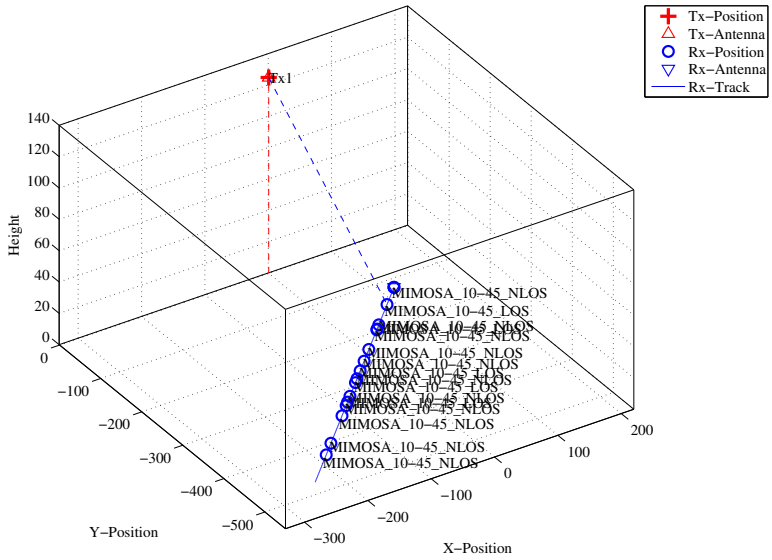

Figure 18: Scenario setup for the comparison of simulated and measured data

Results First, we plot the PDP vs distance from the start point (see Fig. 19).

```
h = cn(1).fr( 20e6,256 );
pdp = squeeze(sum(sum( abs(ifft(h,[],3)).^2 , 1),2));
pdp = 10*log10(pdp.');

figure
imagesc(pdp(end:-1:1,1:192));

cm = colormap('hot');
colormap(cm(end:-1:1,:));

caxis([ max(max(pdp))-60 max(max(pdp))-5 ]);
colorbar

title('Time variant power delay profile');

set(gca,'XTick',1:32:192);
set(gca,'XTickLabel',(0:32:192)/20e6*1e6);
xlabel('Delay [\mus]');

ind = sort( cn.no_snap : -cn(1).no_snap/10 : 1);
set(gca,'YTick', ind );
set(gca,'YTickLabel', round(sort(500-ind / 3,'descend')) );
ylabel('Distance [m]');
```

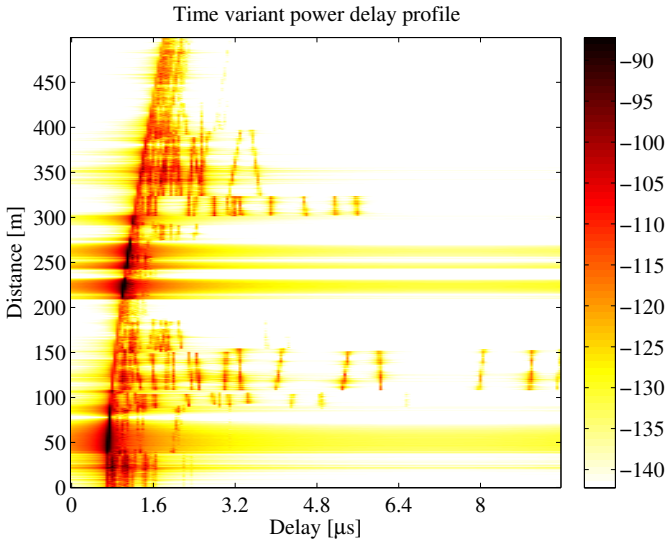

Figure 19: 2D PDP of the simulated track

The next plot shows the total received power along the path (Fig. 20, top left). Green shaded ares are LOS. The rest is NLOS.

```
dist = (1:cn.no_snap)*l.track.get_length/cn.no_snap;
ind  = find(strcmp(l.track.scenario,Sl));
los  = [];
for n = 1:numel(ind)
    los = [los l.track.segment_index(ind(n)) : l.track.segment_index(ind(n)+1)];
end

power = 10*log10( sum( reshape( abs(cn.coeff).^2 , [] , cn.no_snap ) ,1)/4 );
ar = zeros(1,cn.no_snap);
ar(los) = -200;

figure
a = area(dist,ar);
set(a(1),'FaceColor',[0.7 0.9 0.7]);
set(a,'LineStyle','none')

hold on
plot(dist,power)
hold off

title('Position dependent power')
xlabel('Track [m]');
ylabel('Power [dB]');
axis([0 500 min(power)-5 max(power)+5])
legend('LOS','P_{total}',4)
grid on
```

The following plot (Fig. 20, top right) shows the distribution (PDF) of the received power for both, the LOS and NLOS segments.

```
bins   = -150:2:-80;
p_los  = hist(power(los),bins)/cn.no_snap*100;
p_nlos = hist(power(setdiff(1:cn.no_snap,los)),bins)/cn.no_snap*100;

figure
bar(bins,[p_los;p_nlos]')
axis([-124.5,-83,0,ceil(max([p_los,p_nlos]))])
grid on
colormap('Cool')

title('Empirical PDF of the \ac{LOS} and NLOS power')
xlabel('P_{total} [dB]');
ylabel('Probability [%]');
legend('LOS','NLOS',1)
```

The next plot shows the RMS delay spread along the path. Again, shaded ares are for the LOS segments.

```
pow_tap = squeeze(sum(sum(abs(cn.coeff).^2,1),2));
pow_sum = sum( pow_tap,1 );
mean_delay = sum( pow_tap.*cn.delay ,1) ./ pow_sum;
ds = sqrt( sum( pow_tap.*cn.delay.^2 ,1)./ pow_sum - mean_delay.^2 );
ar = zeros(1,cn.no_snap);
ar(los) = 10;

figure
a = area(dist,ar);
set(a(1),'FaceColor',[0.7 0.9 0.7]);
set(a,'LineStyle','none')

hold on
plot( dist , ds*1e6  )
hold off

ma = 1e6*( max(ds)+0.1*max(ds) );
axis([0 500 0 ma])
title('Position dependant delay spread');
xlabel('Track [m]');
ylabel('Delay Spread [dB]');
legend('LOS','\sigma_\tau',1)
grid on
```

The final plot (Fig. 20, bottom right) shows the distribution (PDF) of the RMS delay spread for both, the LOS and NLOS segments.

```
bins = 0:0.03:3;
ds_los  = hist(ds(los)*1e6,bins)/cn.no_snap*100;
ds_nlos = hist(ds(setdiff(1:cn.no_snap,los))*1e6,bins)/cn.no_snap*100;

figure
bar(bins,[ds_los;ds_nlos]')
axis([0,1.5,0,ceil(max([ds_los,ds_nlos]))])
grid on
colormap('Cool')

title('Empirical PDF of the LOS and NLOS RMSDS')
xlabel('\sigma_\tau [\mus]');
ylabel('Probability [%]');
legend('LOS','NLOS',1)
```

```
close all
disp(['QuaDRiGa Version: ',simulation_parameters.version])
```

```
QuaDRiGa Version: 1.0.1-145
```

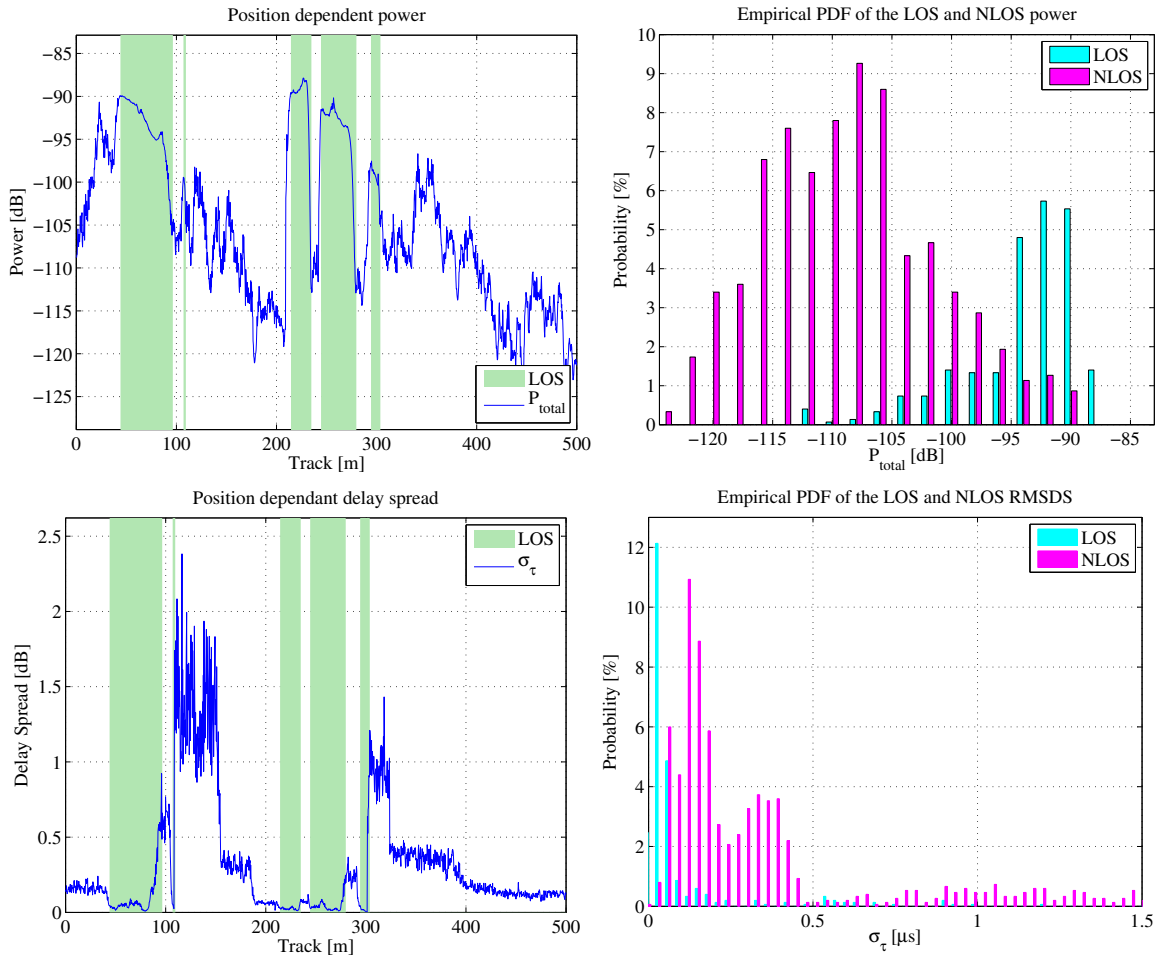

Figure 20: Results for the measurement based simulation tutorial

A.3 Generation of Satellite Channels

This script demonstrates the parametrization of the channel model to generate time-continuous sequences for a satellite scenario.

Setting up the Simulation Parameters First, we set up the general simulation parameters. We choose a center frequency of 2.1 GHz. We also want to use drifting in order to get the correct delays and angles for the time-continuous simulation. A sample density of 2.5 ensures that the channel coefficients can be interpolated to different playback speeds later on.

```
close all
clear all

s = simulation_parameters;          % Basic simulation parameters
s.center_frequency      = 2.185e9;
s.sample_density        = 0.25;

RandStream.setGlobalStream(RandStream('mt19937ar','seed',1));
```

Creating a random Track and defining states along the track Next, we generate a simulation track. A track describes the movement of a mobile terminal. It is composed of an ordered list of positions. During the simulation, one snapshot is generated for each position on the track. Later on, the generation of the track is done by the state sequence generator. Here, we implement a simple version of the sequence generator to generate a random track. We first create a set of streets with different length. We assume a normal distribution of the street length where the parameters mu and sigma were fitted from random distances between two crossings in central Berlin (measured with Google earth).

```
street_length_mu = 187;          % Average street length in m
street_length_sigma = 83;
min_street_length = 50;

turn_probability = 0.4;          % The prob. that the car turns at a crossing
curve_radius = 10;               % The curve radius in m
diro = rand * 2*pi;              % Random start direction
```

For the given parameters, we calculate a list of points along the track that resemble the street grid and the turns at crossings.

```
point = 0;                       % The start point (always at [0,0])
m = 1;                           % A counter for the points
for n = 1:3                      % We simulate 3 street segments

    % Get a random street length drawn from the distribution defined above
    street_length = randn*street_length_sigma + street_length_mu;
    while street_length < min_street_length
        street_length = randn*street_length_sigma + street_length_mu;
    end

    % Get 3 points along the street
    point(m+1) = point(m) + exp(1j*diro) * street_length*0.1;
    point(m+2) = point(m) + exp(1j*diro) * street_length*0.9;
    point(m+3) = point(m) + exp(1j*diro) * street_length;
    m=m+3;

    % At a crossing, the car could change its direction. This is
    % modeled here
    if rand < turn_probability
        dirn = diro + sign( rand-0.5 ) * pi/2 + randn*pi/12;
        point(m+1) = point(m) + curve_radius*( exp(1j*diro) + exp(1j*dirn) );
        diro = dirn;
        m=m+1;
    end
end
```

Next, we create a track object and pass the points along the track. We then use the internal interpolation functions to interpolate the track to 1 point per meter.

```
t = track;                          % Create a track object
t.positions = [ real(point) ; imag(point) ; zeros(1,numel(point))];
t.interpolate_positions( 1 );    % Interpolate to 1 point per meter
```

We now assemble a rudimentary state sequence generator that generates different states along the track. We first define the distribution parameters of the segment length and then calculate the segments themselves. The two possible states are "MIMOSA_10-45_LOS" which stands for LOS or good state and "MIMOSA_10-45_NLOS" for NLOS or bad state.

```
segment_length_mu = 30;          % Average segment length in m
segment_length_sigma = 12;       % Standard deviation in m
min_segment_length = 10;         % Minimum segment length in m

% Now we define the segments (the states) along the track
ind = 1;
while ind < t.no_snapshots

    % Each scenario has a 50% probability
    if rand < 0.5
        t.scenario{ t.no_segments } = 'MIMOSA_10-45_LOS' ;
    else
        t.scenario{ t.no_segments } = 'MIMOSA_10-45_NLOS' ;
    end

    % Get the length of the current segment
    segment_length = randn*segment_length_sigma + segment_length_mu;
    while segment_length<min_segment_length
        segment_length = randn*segment_length_sigma + segment_length_mu;
    end
    segment_length = round(segment_length);      % Segment length
    ind = ind + segment_length;                  % Start of next segment

    if ind < t.no_snapshots      % Exception for the last segment
        t.no_segments = t.no_segments + 1;
        t.segment_index( t.no_segments ) = ind;
    end
end
```

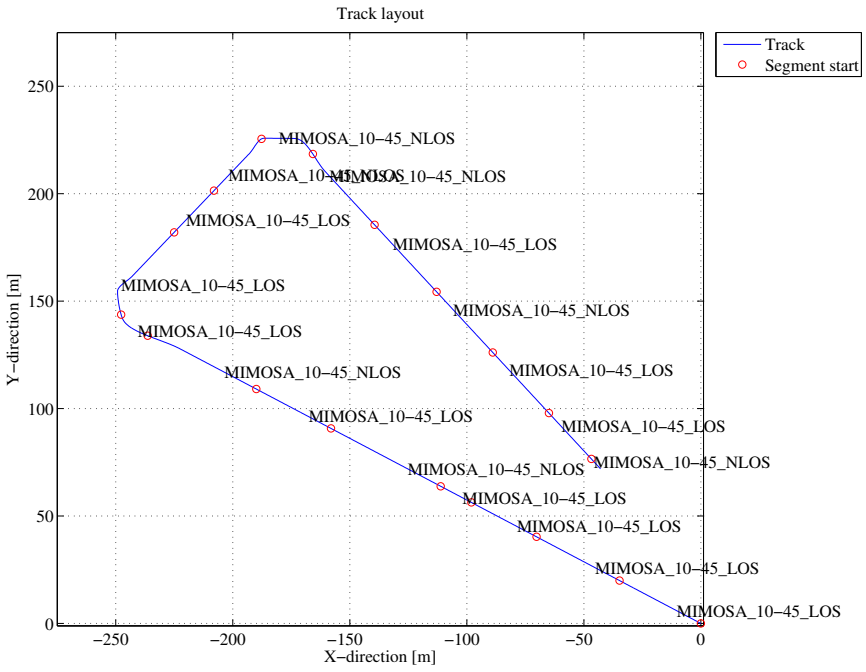

Figure 21: Receiver track for the satellite channel tutorial

Finally, we interpolate the track to the given sample density (2 samples per half-wave-length) and plot the track (see Fig. 21).

```
t.interpolate_positions( s.samples_per_meter );
t.visualize;
```

Defining Antenna Arrays In the third step, we set up our antenna arrays for the transmitter at the satellite and the receiver. We use synthetic dipole antennas for this case. Two dipoles are crossed by an angle of 90 degree. The signal is then split and fed with a 90 degree phase shift to both elements generating RHCP and LHCP signals.

```
% Create a patch antenna with 120 degree opening
a = array('custom',120,120,0);

% Copy element 1 to element 2 - the resulting antenna array has two
% elements, both dipoles.
a.copy_element(1,2);

% Rotate the second pattern by 90 degree around the x-axis.
a.rotate_pattern(90,'x',2);

% Set the coupling between the elements. The Tx-signal for the first
% element is shifted by +90 degree out of phase and put on the second element.
% The signal for the second element is shifted by -90 degree and copied to the
% first element. Both antennas thus radiate a RHCP and a LHCP wave.
a.coupling = 1/sqrt(2) * [1 1;1j -1j];

% Create a copy of the array for the receiver.
b = a.copy_objects;
b.coupling = 1/sqrt(2) * [1 1;1j -1j];

% Rotate the receive antenna array to face sky-wards.
b.rotate_pattern(-90,'y');

b.visualize;                    % Plot the pattern of the Rx-Antenna
```

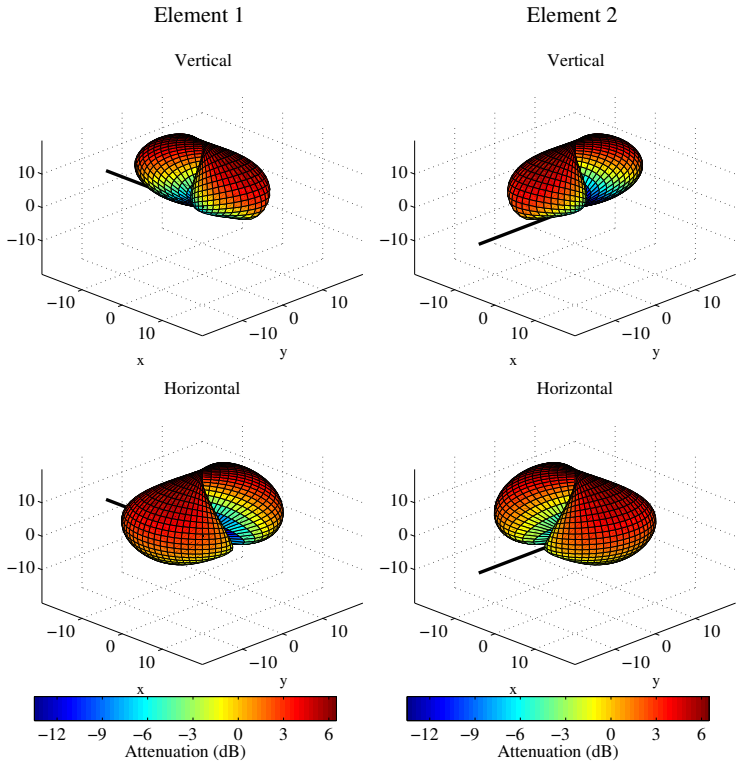

Figure 22: Antenna patterns for the satellite channel tutorial

Setting up the Layout In this step, we combine the track, the antennas and the position of the satellite into a simulation layout. A layout object contains all the geometric information that are necessary to run the simulation. First, we define the position of the satellite. Since the model uses Cartesian coordinates, we have to transform the position of the satellite first.

```
l = layout( s );                % Create a new layout

% Choose a random satellite position (Astra 2, seen from Berlin).
% The distance only needs to be big enough to ensure insignificant changes
% in the reception angle on the ground.

sat_el      = 28.4;             % Elevation angle
sat_az      = 161.6;            % Azimuth angle (South = 180 degree)
rx_latitude = 51;               % Latitude of the Rx

% Approximate the satelite distance for GEO orbit
dist_x      = 35786 + rx_latitude/90 * 6384;    % [km]
dist_y      = (1-rx_latitude/90) * 6384;        % [km]
sat_dist    = sqrt(dist_x^2 + dist_y^2);        % [km]
sat_dist    = sat_dist*1e3;                     % [m]

% Transform angles to Cartesian coordinates
sat_x = sat_dist * cosd(sat_el) * cosd( -sat_az+90 );
sat_y = sat_dist * cosd(sat_el) * sind( -sat_az+90 );
sat_z = sat_dist * sind(sat_el);

% We also turn the antenna of the satellite so it points to the receiver.
a.rotate_pattern( sat_el , 'y' );
a.rotate_pattern( 270-sat_az , 'z' );

% Set the satellite position in the layout
l.tx_position = [ sat_x ; sat_y ; sat_z ];

l.track = t;                    % Set the track for the receiver
l.tx_array = a;                 % Set the tx_array
l.rx_array = b;                 % Set the rx_array
```

Setting up scenario parameters Next, the large scale parameters are set. The first line calls "l.create_parameter_sets", a built-in function that processes the data in the layout and returns a new "parameter_set" object "p". "p" is an array with two elements. One of them contains all the parameters for the good state (LOS) and one for the bad state (NLOS).

```
p = l.create_parameter_sets(0);
```

Each parameter set has two different kinds of parameters. One for the scenario and one for the current state. For example, a scenario might have an average RMS Delay spread of 158 ns plus a certain variance which defines a range for the RMSDS. In addition to that, there are cross-correlations with other parameters such as the angular spread at the transmitter. All those parameters are stored in the "scenpar" property. For the good state, that parameters are:

```
Sl = strcmp( { p(1).name , p(2).name } ,'MIMOSA-10-45-LOS_Tx1' );  % Select good state
p(Sl).scenpar                                                      % Show parameter list
```

```

ans =

           NumClusters: 8
                   r_DS: 2.5000
         PerClusterAS_D: 6.2000e-07
         PerClusterAS_A: 12
         PerClusterES_D: 1.9000e-07
         PerClusterES_A: 7
    LOS_scatter_radius: 0.1000
               LNS_ksi: 3
                xpr_mu: 11.9000
             xpr_sigma: 5.5000
```

```
                DS_mu: -7.5000
             DS_sigma: 0.3000
                AS_D_mu: -4.6000
          AS_D_sigma: 0.1000
             AS_A_mu: 1.5000
          AS_A_sigma: 0.2000
             ES_D_mu: -5.1200
          ES_D_sigma: 0.1000
             ES_A_mu: 1.4000
          ES_A_sigma: 0.1000
            SF_sigma: 3.6000
               KF_mu: 15.5000
            KF_sigma: 5.9000
           DS_lambda: 30.5000
        AS_D_lambda: 1000
        AS_A_lambda: 31.5000
        ES_D_lambda: 1000
        ES_A_lambda: 6
          SF_lambda: 35
          KF_lambda: 4.5000
             asD_ds: 0
             asA_ds: 0.6100
             asA_sf: 0.5600
             asD_sf: 0
              ds_sf: 0.4300
            asD_asA: 0
             asD_kf: 0
             asA_kf: -0.4400
              ds_kf: -0.4600
              sf_kf: -0.3000
             esD_ds: 0
             esA_ds: -0.0500
             esA_sf: 0.1800
             esD_sf: 0
            esD_esA: 0
            esD_asD: 0
            esD_asA: 0
            esA_asD: 0
            esA_asA: 0.1500
             esD_kf: 0
             esA_kf: -0.0300
```

Note that the values are given for a log-normal distribution. Thus, the RMSDS in nanoseconds follows from

```
10^( p(Sl).scenpar.DS_mu ) * 1e9
```

```

ans =

   31.6228
```

Each parameter on that list can be changed by just assigning it a new value. Here, we set the number of clusters for the LOS scenario to 7. Note that the default settings are stored in files in the sub-folder "config" of the channel model folder. Here, the default settings can be permanently set. After a change, the parameters of the segments need to be updated. This is done by calling the "update_parameters" method.

```
p(Sl).scenpar.NumClusters = 7;
p.update_parameters;
```

```
Parameters   [oooooooooooooooooooooooooooooooooooooooooooooooooo]    24 seconds
```

When "update_parameter" is called, the specific parameters for each segment are generated. E.g. each segment gets assigned a RMS Delay Spread and other values which are drawn from the statistics defined in scenpar. For the LOS segments, the individual RMSDS values for each segment are:

```
rmsds = p(Sl).ds*1e9
average = mean(p(Sl).ds*1e9)
```

```
rmsds =

  Columns 1 through 7

   48.8391    6.8370   55.2154   48.7273   25.9658   29.7929   22.2436

  Columns 8 through 11

   68.7237   13.3159   85.5425  121.5296


average =

   47.8848
```

Generate channel coefficients Next, we generate the channel coefficients. This is a lengthy task. The next line then combines the channels of the individual segments into a time-continuous channel. Here, the parameter (0.2) sets the length of the overlap region between two segments. In this case, it is 20%.

```
c = p.get_channels;                                % Generate coefficients
cn = c.merge(0.2);                                 % Combine segments
```

```
Channels      [ooooooooooooooooooooooooooooooooooooooooooooooooooo]    21 seconds
Merging       [ooooooooooooooooooooooooooooooooooooooooooooooooooo]     1 seconds
```

Evaluation of the data The next two plots show some basic evaluations of the generated coefficients. The first plot shows the received power for the 4 MIMO links along the track. The plot shows the differences between the LOS and NLOS segments and the cross-pol discrimination between the MIMO links. The average path loss for LOS was set to -95 dB and for NLOS -113 dB.

```
dist = (1:cn.no_snap)*t.get_length/cn.no_snap;
ind  = find(strcmp(t.scenario,'MIMOSA_10-45_LOS'));
los  = [];
for n = 1:numel(ind)
    start = t.segment_index(ind(n));
    if n==numel(ind)
        try
            stop =  t.segment_index(ind(n)+1);
        catch
            stop = t.no_snapshots;
        end
    else
        stop =  t.segment_index(ind(n)+1);
    end
    los = [los start:stop];
end

power = reshape( 10*log10( squeeze(sum( abs(cn.coeff).^2 , 3 )) ) , 4,[]);

mi = min(reshape(power,[],1)) - 5;
ma = max(reshape(power,[],1)) + 5;

ar = ones(1,cn.no_snap) * ma;
ar(los) = mi;

figure('Position',[ 100 , 100 , 1000 , 700]);
a = area(dist,ar);
set(a(1),'FaceColor',[0.7 0.9 0.7]);
set(a,'LineStyle','none')

hold on
plot(dist,power')
hold off

```

```
xlabel('Track [m]');
ylabel('Received Power per MIMO LINK [dB]');
axis([0 t.get_length mi ma])
legend('LOS','P_{11}','P_{12}','P_{21}','P_{22}',4)
box on

title('Received power along the track')
```

The next plot shows the RMS delay spread along the path for the first MIMO element. Again, shaded ares are for the LOS segments.

```
pow_tap = abs(squeeze(cn.coeff(1,1,:,:))).^2;
pow_sum = sum( pow_tap,1 );
mean_delay = sum( pow_tap.*cn.delay ,1) ./ pow_sum;
ds = sqrt( sum( pow_tap.*cn.delay.^2 ,1)./ pow_sum - mean_delay.^2 );
ar = zeros(1,cn.no_snap);
ar(los) = 10000;

figure('Position',[ 100 , 100 , 1000 , 700]);
a = area(dist,ar);
set(a(1),'FaceColor',[0.7 0.9 0.7]);
set(a,'LineStyle','none')

hold on
plot( dist , ds*1e9  )
hold off

ma = 1e9*( max(ds)+0.1*max(ds) );

axis([0 t.get_length 0 ma])
xlabel('Track [m]');
ylabel('Delay Spread [ns]');
legend('LOS','\sigma_\tau',1)
title('Position dependant delay spread');
```

```
close all
disp(['QuaDRiGa Version: ',simulation_parameters.version])
```

```
QuaDRiGa Version: 1.0.1-145
```

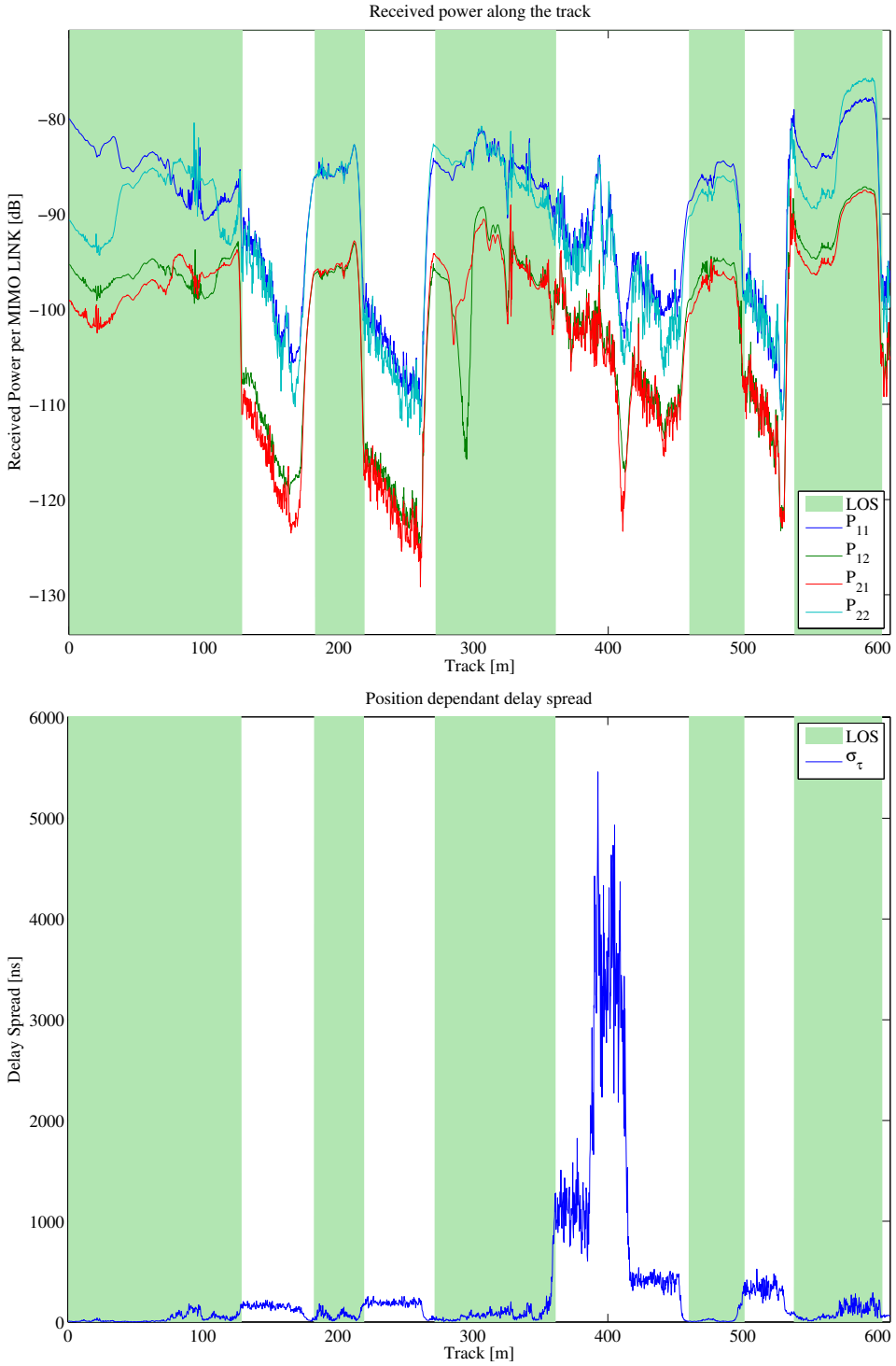

Figure 23: Results for the satellite channel tutorial

A.4 Drifting Phases and Delays

Drifting is an essential feature of the channel model. Drifting enables a continuous time evolution of the path delays, the path phases, the departure- and arrival angles and the LSPs. It is thus the enabling feature for time continuous channel simulations. Although drifting was already available in the SCME branch of the WINNER channel model, it did not make it into the main branch. Thus, drifting is not available in the WIM1, WIM2 or WIM+ model. Here the functionality is implemented again. This script focuses on the delay and the phase component of the drifting functionality.

Channel model setup and coefficient generation First, we parameterize the channel model. We start with the basic simulation parameters. For the desired output, we need two additional options: we want to evaluate absolute delays and we need to get all 20 subpaths. Normally, the subpaths are added already in the channel builder.

```
s = simulation_parameters;
s.center_frequency = 2.53e9;
s.sample_density = 4;
s.use_subpath_output = 1;
s.use_absolute_delays = 1;
```

Second, we define a user track. Here we choose a linear track with a length of 30 m. The track start 20 m east of the transmitter and runs in east direction, thus linearly increasing the distance from the receiver.

```
l = layout( s );
l.tx_position(3) = 25;
l.track.generate('linear',30,0);
l.track.initial_position = [20;0;0];
l.track.scenario = 'WINNER_UMa_C2_LOS';
l.track.interpolate_positions( s.samples_per_meter );
l.visualize;
```

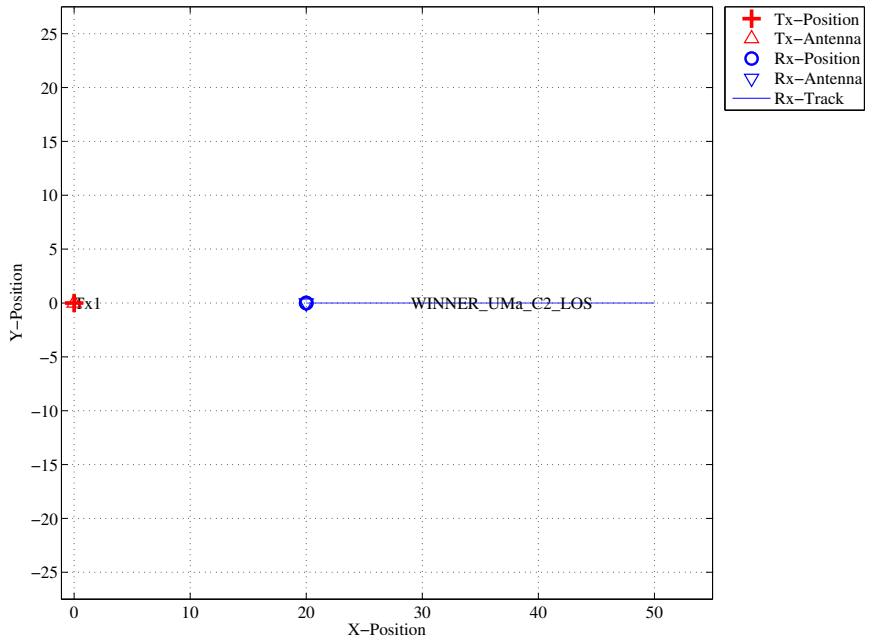

Figure 24: Scenario setup for the drifting phases tutorial

Now, we generate the LSPs. In order to get repeatable results, we set a specific random seed. This is a MATLAB internal function and is not a feature of the channel model. We also set the shadow fading and K-factor to 1 and disable the path loss model.

```
RandStream.setGlobalStream(RandStream('mt19937ar','seed',5));
p = l.create_parameter_sets;
p.parameter_maps(:,:,[2 3]) = 0;            % Fix SF and KF to 0
p.plpar = [];                               % Disable path loss model
p.update_parameters;
```

```
Parameters    [ooooooooooooooooooooooooooooooooooooooooooooooooooo]     1 seconds
```

Now, we generate the channel coefficients. The first run uses the new drifting module, the second run disables it. Note that drifting needs significantly more computing resources. In some scenarios it might thus be useful to disable the feature to get quicker simulation results.

```
s.drifting_precision = 1;
RandStream.setGlobalStream(RandStream('mt19937ar','seed',2));
c = p.get_channels;

s.drifting_precision = 0;
RandStream.setGlobalStream(RandStream('mt19937ar','seed',2));
d = p.get_channels;
```

```
Channels      [ooooooooooooooooooooooooooooooooooooooooooooooooooo]    12 seconds
Channels      [ooooooooooooooooooooooooooooooooooooooooooooooooooo]     7 seconds
```

Results and discussion The following plots represent the results of the test.

```
figure
distance = 20+(1:c.no_snap)*l.track.get_length/c.no_snap;
plot( distance, c.delay(1,:)*1e9 , '-b' )
hold on
plot( distance, d.delay(1,:)*1e9 , '-.b' )
plot( distance, c.delay(2,:)*1e9 , '-r' )
plot( distance, d.delay(2,:)*1e9 , '-.r' )
hold off
xlabel('Distance from track start point')
ylabel('Delay [ns] ')
```

The first plot (Fig. 25) shows the delay of the LOS tap (blue) and the delay of the first NLOS tap (red) vs. distance. The solid lines are from the channel with drifting, the dashed lines are from the channel without. The LOS delay is always increasing since the Rx is moving away from the Tx. However, the increase is not linear due to the 25 m height of the Tx. Without drifting, the delays are not updated and stay constant during the segment. The position of the first scatterer is in close distance to the Rx (only some m away).

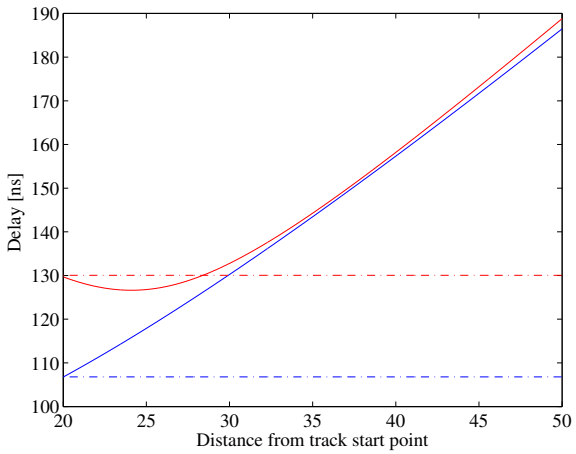

Figure 25: Cluster delays vs. Rx position (drifting phases tutorial)

When moving, the Rx first approaches the scatterer (delay gets a bit smaller) and then the distance increases again.

```
phase = unwrap(angle(squeeze((c.coeff(1,1,2,:,:)))));
plot( distance,phase )
xlabel('Distance from track start point')
ylabel('Continuous phase')
```

The second plot (Fig. 26, left) shows the phases of the 20 subpaths of the first NLOS tap for the drifting case. Note that the phases are not linear. This comes from the close proximity of the scatterer to the initial Rx position. The position of all 20 reflection points are calculated by the channel model. Those position mark the position of the last bounce scatterer (LBS). When moving the Rx, the distance to the LBS changes for each subpath and so does the phase. Here, the phase of each of the subpaths is calculated from the length of the path.

```
pow = abs(squeeze(sum( c.coeff(1,1,2,:,:) , 5 ))).^2;
plot( distance,10*log10(pow),'-r' )
xlabel('Distance from track start point')
ylabel('Tap power (dB)')
```

This plot shows the power of the first NLOS tap along the track. The fading is significantly higher in the beginning and becomes much less strong towards the end.

```
phase = unwrap(angle(squeeze((d.coeff(1,1,2,:,:)))));
plot( distance,phase )
xlabel('Distance from track start point')
ylabel('Continuous phase')
```

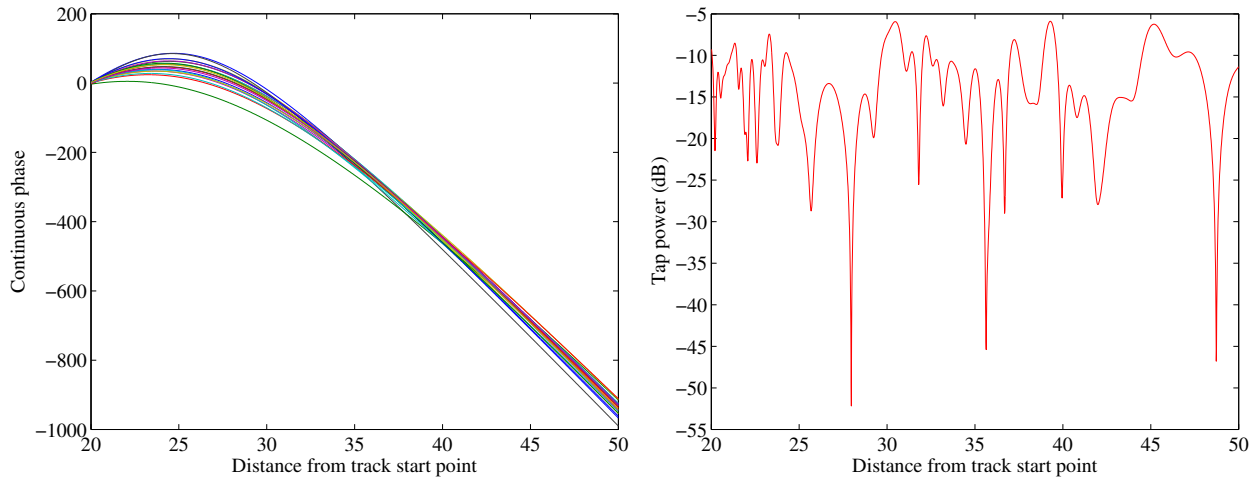

Figure 26: Drifting phases and Tx power vs. Rx position (drifting phases tutorial)

Without drifting, the phases of the subpaths are approximated by assuming that the angles to the LBSs do not change. However, this only holds when the distance to the LBS is large. Here, the initial distance is small (ca. 5 m). When the initial angles are kept fixed along the track, the error is significant. Here, the phase ramp is negative, indicating a movement direction towards the scatterer and thus a higher Doppler frequency. However, when the scatterer is passed, the Rx moves away from the scatterer and the Doppler frequency becomes lower. This is not reflected when drifting is turned off.

Note that with shorter delay spreads (as e.g. in satellite channels), the scatterers are placed closer to the Rxs initial position. This will amplify this effect. Hence, for correct time evolution results, drifting needs to be turned on.

eMail: quadriga@hhi.fraunhofer.de

```
pow = abs(squeeze(sum( d.coeff(1,1,2,:,:) , 5 ))).^2;
plot( distance,10*log10(pow),'-r' )
xlabel('Distance from track start point')
ylabel('Tap power (dB)')
```

```
close all
disp(['QuaDRiGa Version: ',simulation_parameters.version])
```

```
QuaDRiGa Version: 1.0.1-145
```

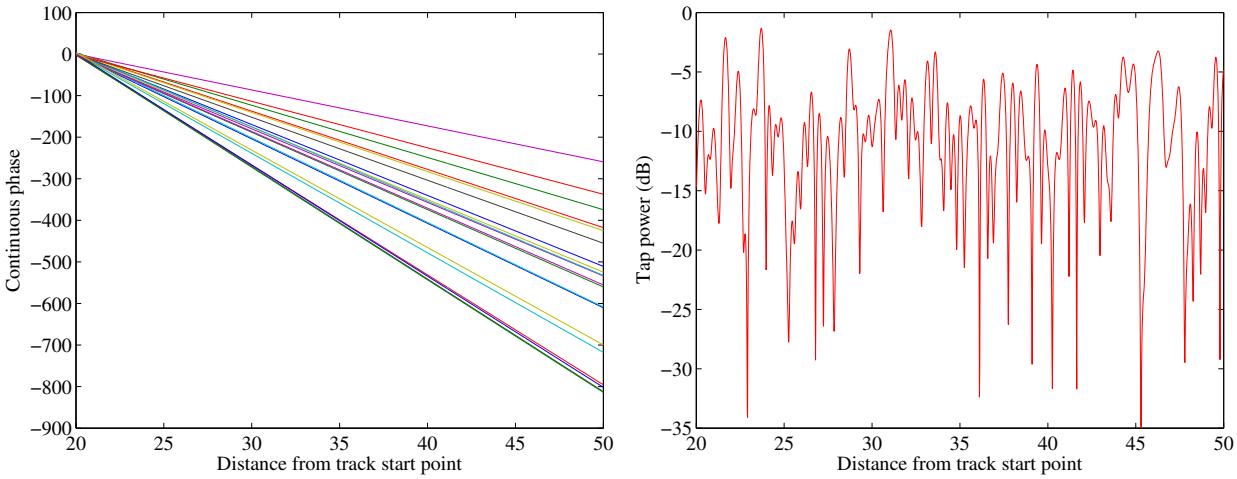

Figure 27: Phases and Tx power vs. Rx position without drifting (drifting phases tutorial)

A.5 Time Evolution and Scenario Transitions

The channel model generates the coefficients separately for each segment. In order to get a time-continuous output, these coefficients have to be combined. This is a feature originally described in the documentation of the WIM2 channel model, although it was never implemented. Since this component is needed for time-continuous simulations, it was implemented here. This script sets up the simulation and creates such time-continuous CIRs.

Channel model setup and coefficient generation First, we set up the channel model.

```
close all
clear all

set(0,'defaultTextFontSize', 14)
set(0,'defaultAxesFontSize', 14)

s = simulation_parameters;
s.center_frequency = 2.53e9;
s.sample_density = 4;
s.use_absolute_delays = 1;
```

Second, we create a more complex network layout featuring an elevated transmitter (25 m) and two receivers at 1.5 m height. The first Rx moves along a circular track around the receiver. The second receiver moves away from the Tx. Both start at the same point.

Note that each track is split into three segments. The first Rx goes from an LOS area to a shaded area and back. The second track also starts in the LOS area. Here, the scenario changes to another LOS segment and then to an NLOS segment. The LOS-LOS change will create new small-scale fading parameters, but the LSPs will be highly correlated between those two segments.

```
l = layout( s );                                % New layout
l.no_rx = 2;                                    % Two receivers

l.tx_array.generate('dipole');                  % Dipola antennas at all Rx and Tx
l.rx_array = l.tx_array;

l.tx_position(3) = 25;                          % Elevate Tx to 25 m

UMal = 'BERLIN_UMa_LOS';
UMan = 'BERLIN_UMa_NLOS';

l.track(1).generate('circular',20*pi,0);        % Circular track with 10m radius
l.track(1).initial_position    = [10;0;1.5];    % Start east, running nord
l.track(1).segment_index       = [1,40,90];     % Segments
l.track(1).scenario            = { UMal , UMan , UMal };

l.track(2).generate('linear',20,0);             % Linear track, 20 m length
l.track(2).initial_position    = [10;0;1.5];    % Same start point
l.track(2).interpolate_positions( 128/20 );
l.track(2).segment_index       = [1,40,90];
l.track(2).scenario            = { UMal , UMal , UMan };

l.visualize;                                    % Plot all tracks

l.track.interpolate_positions( s.samples_per_meter );
l.track.compute_directions;
```

Now we create the channel coefficients. Fixing the random seed guarantees repeatable results (i.e. the taps will be at the same positions for both runs). Note that the computing time is significantly longer when drifting is enabled.

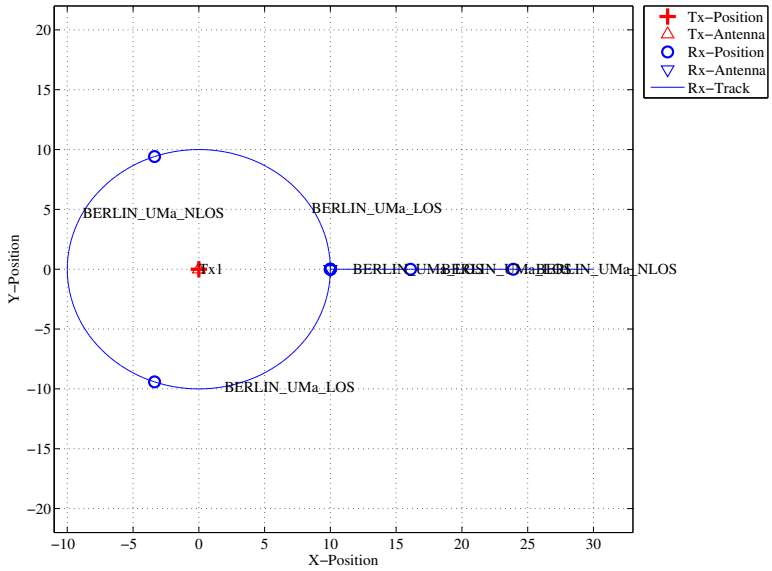

Figure 28: Scenario setup for the time evolution tutorial

```
RandStream.setGlobalStream(RandStream('mt19937ar','seed',2));
p = l.create_parameter_sets;

disp('Drifting enabled:');
s.drifting_precision = 1;
RandStream.setGlobalStream(RandStream('mt19937ar','seed',1));
c = p.get_channels;
cn = c.merge;

disp('Drifting disabled:');
s.drifting_precision = 0;
RandStream.setGlobalStream(RandStream('mt19937ar','seed',1));
d = p.get_channels;
dn = d.merge;
```

```
Parameters   [ooooooooooooooooooooooooooooooooooooooooooooooooooo]     1 seconds
Drifting enabled:
Channels     [ooooooooooooooooooooooooooooooooooooooooooooooooooo]    36 seconds
Drifting disabled:
Channels     [ooooooooooooooooooooooooooooooooooooooooooooooooooo]     5 seconds
```

Results and discussion Now we plot and discuss the results. We start with the power of the LOS tap along the circular track and compare the outcome with and without drifting (Fig. 29, left).

```
degrees = (0:cn(1).no_snap-1)/cn(1).no_snap * 360;
los_pwr_drift = 10*log10(squeeze(abs(cn(1).coeff(1,1,1,:))).^2);
los_pwr_nodrift = 10*log10(squeeze(abs(dn(1).coeff(1,1,1,:))).^2);

figure
plot( degrees,los_pwr_drift )
hold on
plot(degrees,los_pwr_nodrift ,'-.r')
hold off

a = axis;
axis( [0 360 a(3:4) ] );

xlabel('Position on circle [\degree\]');
ylabel('Power of the LOS component');
title('Power of the LOS component for the circular track');
legend('Drifting','No drifting',4);
```

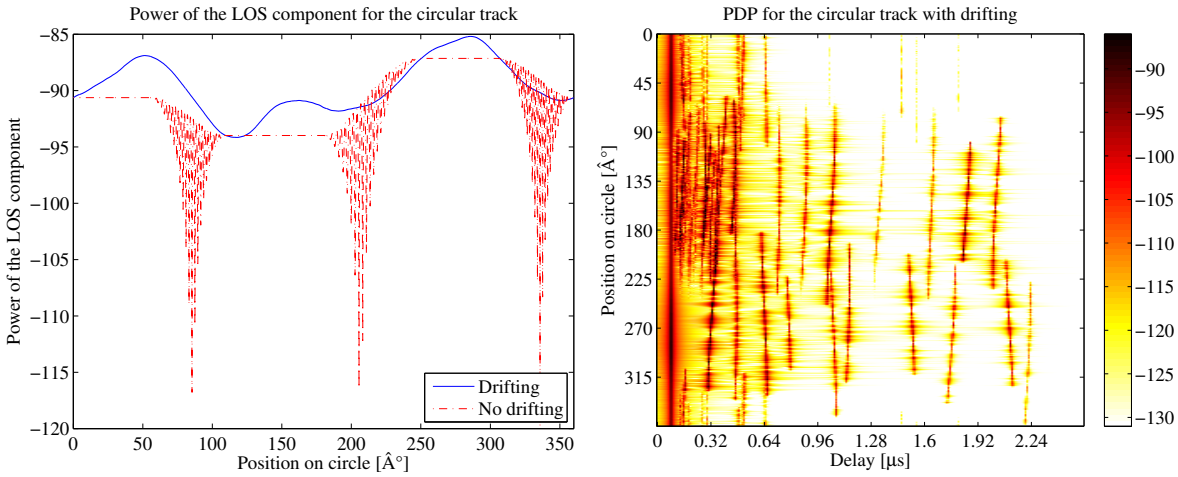

Figure 29: Received power on the circular track (time evolution tutorial)

When drifting is enabled (Fig. 29, left, blue curve), the channel output after merging is time-continuous. The variations along the track come from the drifting K-Factor and the drifting shadow fading. When drifting is disabled, these parameters are not updated and kept fixed at their initial value.

At the end of each segment, both channels are cross-faded. I.e. the power of the output of the first segment ramps down and the power of the second segment ramps up. Since drifting guarantees a time-continuous evolution of the phase, this ramping process is also time continuous and no artifacts are visible in the blue curve.

Without drifting, however, the phases are approximated based on their initial values, the initial arrival- and departure angles and the traveled distance from the start point. However, since the Rx moves along a circular track, the angles change continuously which is not correctly modeled. The phase at the end of the first segment does not match the phase at the beginning of the second. When adding both components, artifacts appear as can be seen in the red curve.

Next, we plot the power-delay profiles for both tracks. We calculate the frequency response of the channel and transform it back to the time domain by an IFFT. Then, we create a 2D image of the received power at each position of the track. We start with the circular track.

```
h = cn(1).fr( 100e6,512 );
h = squeeze(h);
pdp = 10*log10(abs(ifft(h,[],1).').^2);

figure
imagesc(pdp(:,1:256));
caxis([ max(max(pdp))-50 max(max(pdp))-5 ]);
colorbar

cm = colormap('hot');
colormap(cm(end:-1:1,:));

set(gca,'XTick',1:32:255);
set(gca,'XTickLabel',(0:32:256)/100e6*1e6);
xlabel('Delay [\mus]');

set(gca,'YTick',1:cn(1).no_snap/8:cn(1).no_snap);
set(gca,'YTickLabel', (0:cn(1).no_snap/8:cn(1).no_snap)/cn(1).no_snap * 360 );
ylabel('Position on circle [\degree\]');

title('PDP for the circular track with drifting');
```

The X-axis (Fig. 29, right) shows the delay in microseconds and the Y-axis shows the position on the circle. For easier navigation, the position is given in degrees. 0° means east (starting point), 90° means north, 180° west and 270° south. The LOS delay stays constant since the distance to the Tx is also constant.

However, the power of the LOS changes according to the scenario. Also note, that the NLOS segment has significantly more taps due to the longer delay spread. Next, we create the same plot for the linear track (Fig. 30). Note the slight increase in the LOS delay and the high similarity of the first two LOS segments due to the correlated LSPs. Segment change is at around 6 m.

```
h = cn(2).fr( 100e6,512 );
h = squeeze(h);
pdp = 10*log10(abs(ifft(h,[],1).').^2);

figure
imagesc(pdp(:,1:256));
caxis([ max(max(pdp))-50 max(max(pdp))-5 ])
colorbar

cm = colormap('hot');
colormap(cm(end:-1:1,:));

set(gca,'XTick',1:32:255);
set(gca,'XTickLabel',(0:32:256)/100e6*1e6);
xlabel('Delay [\mus]');

set(gca,'YTick',1:cn(2).no_snap/8:cn(2).no_snap);
set(gca,'YTickLabel', (0:cn(2).no_snap/8:cn(2).no_snap)/cn(2).no_snap * 20 );
ylabel('Distance from start point [m]');

title('PDP for the linear track with drifting');
```

Last, we plot the same results for the linear track without drifting (Fig. 30, right). Note that the LOS delay is not smooth during segment change. There are two jumps at 6 m and again at 13.5 m.

```
h = dn(2).fr( 100e6,512 );
h = squeeze(h);
pdp = 10*log10(abs(ifft(h,[],1).').^2);

figure
imagesc(pdp(:,1:256));
caxis([ max(max(pdp))-50 max(max(pdp))-5 ])
colorbar

cm = colormap('hot');
colormap(cm(end:-1:1,:));

set(gca,'XTick',1:32:255);
set(gca,'XTickLabel',(0:32:256)/100e6*1e6);
xlabel('Delay [\mus]');

set(gca,'YTick',1:cn(2).no_snap/8:cn(2).no_snap);
set(gca,'YTickLabel', (0:cn(2).no_snap/8:cn(2).no_snap)/cn(2).no_snap * 20 );
ylabel('Distance from start point [m]');

title('PDP for the linear track without drifting');
```

```
close all
disp(['QuaDRiGa Version: ',simulation_parameters.version])
```

```
QuaDRiGa Version: 1.0.1-145
```

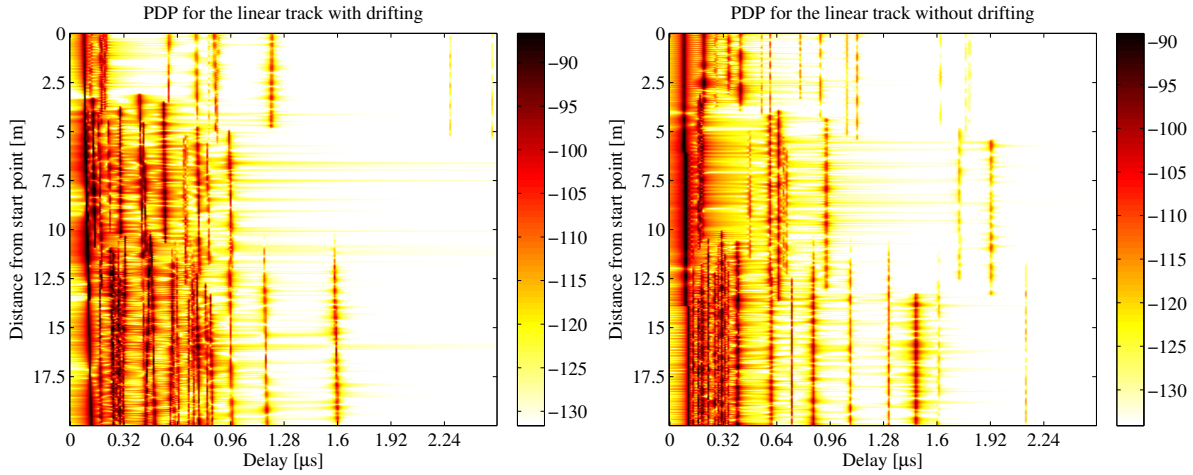

Figure 30: Received power on the linear track (time evolution tutorial)

A.6 Applying Varying Speeds (Channel Interpolation)

One new feature that makes the simulations more realistic is the function to apply arbitrary speed- and movement profiles, e.g. accelerating, breaking or moving at any chosen speed. These profiles are defined in the track class. The profiles are then converted into effective sampling points which aid the interpolation of the channel coefficients.

Channel model setup First, we set up the simulation parameters. Note the sample density of 2.5 which enables very fast simulations even with drifting.

```
s = simulation_parameters;
s.center_frequency = 2.53e9;
s.sample_density = 2.5;
s.use_absolute_delays = 1;
```

Second, we define a track. It has a length of 20 m, starts at 10 m east of the transmitter and consists of three segments (LOS, NLOS, LOS). The positions are interpolated to match the sample density defined above. The track is then plugged into a network layout with one transmitter at position (0,0,25). Both, transmitter and receiver are equipped with dipole antennas. The last three lines create the LSPs.

```
t = track('linear',20,0);
t.initial_position = [60;0;1.5];
t.interpolate_positions( 128/20 );
t.segment_index       = [1,40,90];
t.scenario            = { 'WINNER_UMa_C2_LOS' , 'WINNER_UMa_C2_NLOS' ,...
    'WINNER_UMa_C2_LOS' };
t.interpolate_positions( s.samples_per_meter );

l = layout( s );
l.tx_array.generate('dipole');
l.rx_array = l.tx_array;
l.tx_position(3) = 25;
l.track = t;

l.visualize;

RandStream.setGlobalStream(RandStream('mt19937ar','seed',5));
p = l.create_parameter_sets;
```

```
Parameters   [ooooooooooooooooooooooooooooooooooooooooooooooooooo]     2 seconds
```

Channel generation and results Next, we generate the channel coefficients. Note that the initial sample density is 2.5. We then interpolate the sample density to 20. It would take ten times as long to achieve the same result with setting the initial sample density to 20. The interpolation is significantly faster. It is done by first setting the speed to 1 m/s (default setting) and then creating a distance vector which contains a list of effective sampling points along the track.

```
c = p.get_channels;
cn = c.merge;

t.set_speed( 1 );
dist = t.interpolate_movement( s.wavelength/(2*20) );
ci = cn.interpolate( dist , t.get_length , 'spline' );
```

```
Channels     [ooooooooooooooooooooooooooooooooooooooooooooooooooo]     5 seconds
```

The next plot shows the power of the first three taps from both, the original and the interpolated channel, plotted on top of each other. The values are identical except for the fact, that the interpolated values (blue line) have 5 times as many sample points.

eMail: quadriga@hhi.fraunhofer.de

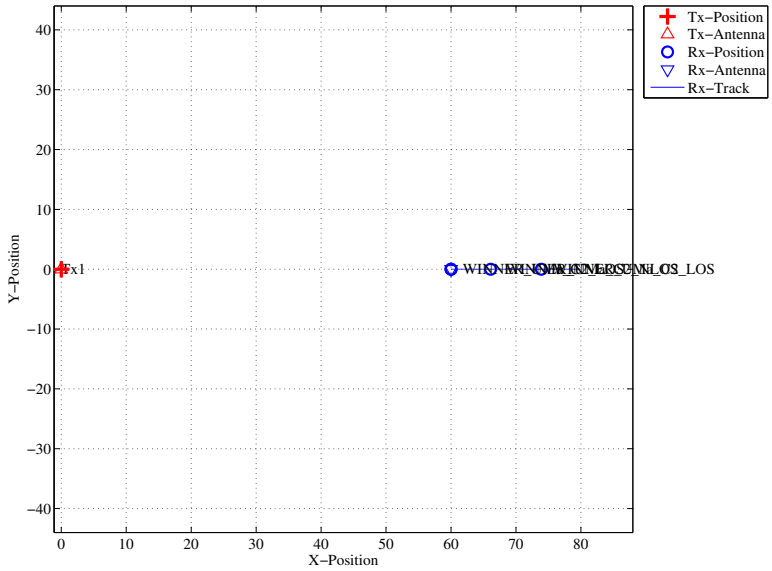

Figure 31: Scenario setup for the speed profile tutorial

```
pwr_orig = 10*log10(squeeze(abs(cn.coeff(1,1,1:3,:))).^2);
nsnap = cn.no_snap;
dist_orig = (0:nsnap-1) * t.get_length/(nsnap-1);
pwr_int = 10*log10(squeeze(abs(ci.coeff(1,1,1:3,:))).^2);

figure
plot( dist_orig,pwr_orig , 'r','Linewidth',2 )
hold on
plot( dist,pwr_int ,'b' )
hold off
axis(  [ min(dist) , max(dist) ,...
    min( pwr_orig( pwr_orig>-Inf ) ) , ...
    max( pwr_orig( pwr_orig>-Inf ) )+10 ] )

xlabel('Distance from start point [m]');
ylabel('Power [dB]');
```

Fig. 32 (right) shows the power delay profile (PDP) for the interpolated channel. As defined in the track object, it starts with a LOS segment, going into a shaded area with significantly more multipath fading at around 4 seconds and then back to LOS at around 13 sec.

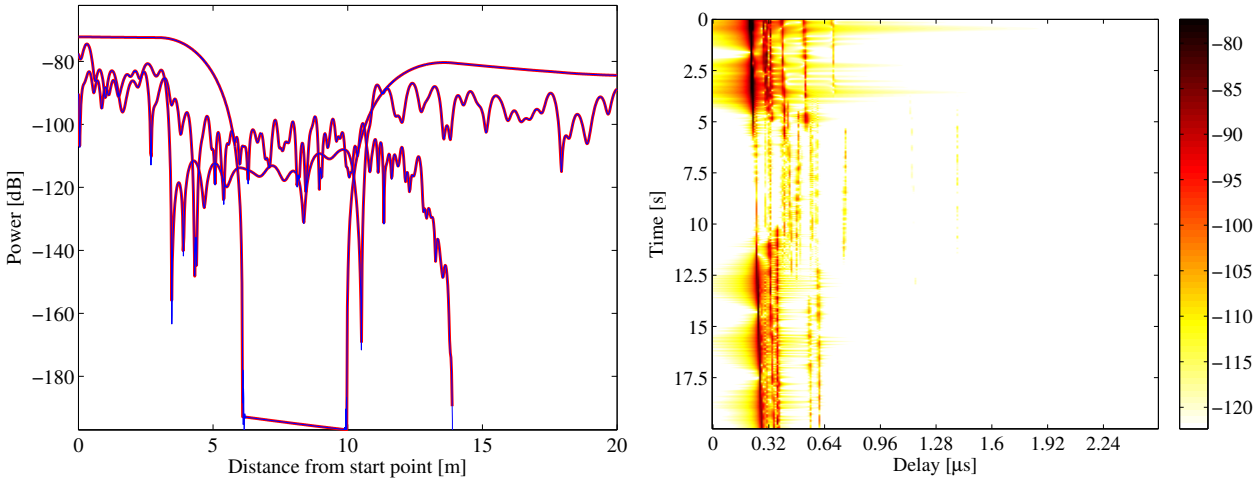

Figure 32: Received power and 2D PDP for the speed profile tutorial

```
h = ci.fr( 100e6,512 );
h = squeeze(h);
pdp = 10*log10(abs(ifft(h,[],1).').^2);

figure
imagesc(pdp(:,1:256));
caxis([ max(max(pdp))-50 max(max(pdp))-5 ])
colorbar

cm = colormap('hot');
colormap(cm(end:-1:1,:));

set(gca,'XTick',1:32:255);
set(gca,'XTickLabel',(0:32:256)/100e6*1e6);
xlabel('Delay [\mus]');

set(gca,'YTick',1:ci.no_snap/8:ci.no_snap);
set(gca,'YTickLabel', (0:ci.no_snap/8:ci.no_snap)/ci.no_snap * 20 );
ylabel('Time [s]');
```

Now, we create a movement profile. It is defined by a set of value pairs in track.movement_profile. The first value represents the time in seconds, the second value the position on the track. Here, we start at a position of 7 m, i.e. in the second (NLOS) segment. We then go back to the beginning of the track. This takes 5 seconds. Then, we wait there for 1 second and go to the end of the track, which we reach after additional 14 seconds.

The next step is to interpolate the sample points. This is done by the interpolate_movement method. It requires the sample interval (in s) as an input argument. Here, we choose an interval of 1 ms which gives us 1000 samples per second. Fig. 33 (left) illustrates the results.

```
t.movement_profile = [ 0,7 ; 5,0 ; 6,0 ; 20,20  ]';
dist = t.interpolate_movement( 1e-3 );
ci = cn.interpolate( dist , t.get_length );

nsnap = ci.no_snap;
time = (0:nsnap-1) * t.movement_profile(1,end)/(nsnap-1);

figure
plot( time,dist , 'r' )

xlabel('Time [s]');
ylabel('Position on track [m]');
```

The last plot (Fig. 33, right) shows the PDP of the interpolated channel with the movement profile applied. The channel starts in the second segment with a lot of fading, goes back to the first while slowing down at the same time. After staying constant for one second, the channel starts running again, speeding up towards the end of the track.

```
h = ci.fr( 100e6,512 );
h = squeeze(h);
pdp = 10*log10(abs(ifft(h,[],1).').^2);

figure
imagesc(pdp(:,1:256));
caxis([ max(max(pdp))-50 max(max(pdp))-5 ])
colorbar

cm = colormap('hot');
colormap(cm(end:-1:1,:));

set(gca,'XTick',1:32:255);
set(gca,'XTickLabel',(0:32:256)/100e6*1e6);
xlabel('Delay [\mus]');

set(gca,'YTick',1:ci.no_snap/8:ci.no_snap);
set(gca,'YTickLabel', (0:ci.no_snap/8:ci.no_snap)/ci.no_snap * 20 );
ylabel('Time [s]');
```


eMail: quadriga@hhi.fraunhofer.de

```
close all
disp(['QuaDRiGa Version: ',simulation_parameters.version])
```

```
QuaDRiGa Version: 1.0.1-145
```

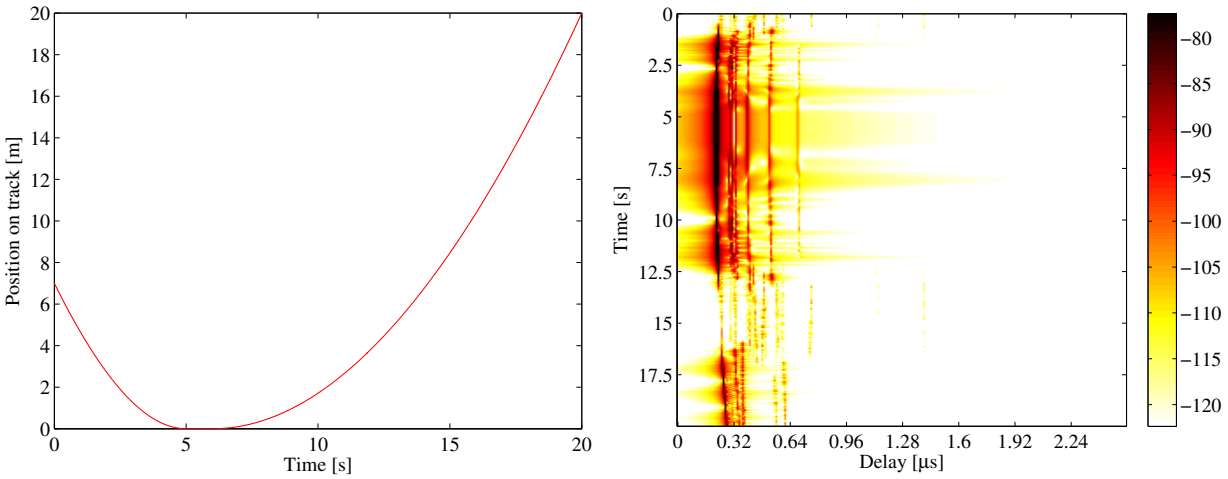

Figure 33: Movement profile (left) and interpolated PDP (right)

A.7 Geometric Polarization

Here, we demonstrate the polarization rotation model that calculates the path power for polarized antenna arrays. We do this by setting up the simulation with different H/V polarized antennas at the transmitter and at the receiver. Then we define a circular track around the receiver. When the receiver moves around the transmitter, it changes its antenna orientation according to the movement direction. In this way, all possible departure and elevation angles are sampled. Depending on the antenna orientation, the polarizations are either aligned (e.g. the Tx is V-polarized and the Rx is V-polarized), they are crossed (e.g. the Tx is V-polarized and the Rx is H-polarized) or the polarization orientation is in between those two. The generated channel coefficients should reflect this behavior.

Setting up the simulation environment First, we have to set up the simulator with some default settings. Here, we choose a center frequency of 2.1 GHz. We also want to use drifting in order to get the correct angles for the LOS component and we set the number of transmitters and receivers to one.

```
close all
clear all

set(0,'defaultTextFontSize', 14)
set(0,'defaultAxesFontSize', 14)

s = simulation_parameters;                  % Set the simulation parameters
s.center_frequency = 2.1e9;                 % Center-frequency: 2.1 GHz
s.use_polarization_rotation = 1;
s.samples_per_meter = 360/(40*pi);          % One sample per degree
s.drifting_precision = 1;
```

Setting up the antenna arrays In the second step, we set up our antenna arrays. We use the synthetic dipole antennas for this case. Those antennas show perfect polarization characteristics. First, we generate a

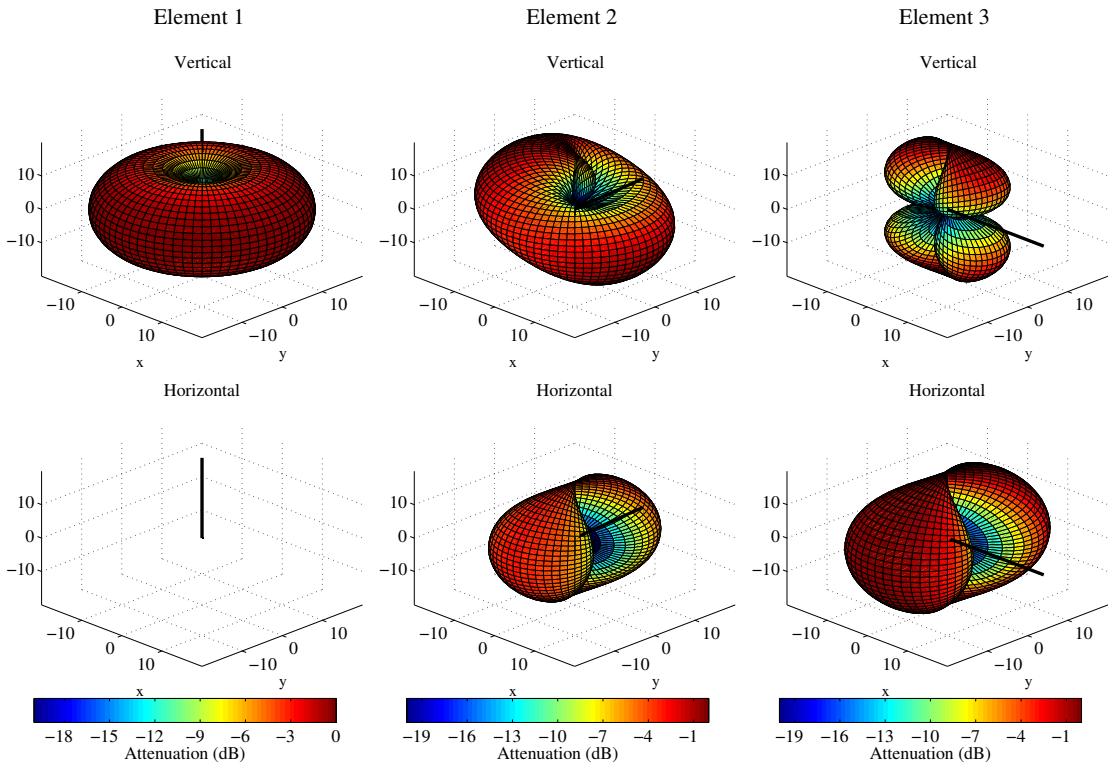

Figure 34: Polarimetric dipole antenna patterns for different orientations

single dipole with V-polarization. Then, we create multiple copies of this antenna element and rotate them by 45 and 90 degrees, respectively. We then use the same array for the receiver.

```
l = layout( s );                                % Create a new Layout
l.tx_array.generate('dipole');                  % create V-polarized dipole
l.tx_array.set_grid( (-180:10:180)*pi/180 , (-90:10:90)*pi/180 );

l.tx_array.field_pattern_vertical = ...         % Normalize
    l.tx_array.field_pattern_vertical / max(max(l.tx_array.field_pattern_vertical));

l.tx_array.set_grid( (-180:5:180)*pi/180 , (-90:5:90)*pi/180 );
l.tx_array.copy_element(1,2:3);                 % Duplicate the element two more times
l.tx_array.rotate_pattern(45,'y',2);            % 45\degree\ polarization
l.tx_array.rotate_pattern(90,'y',3);            % 90\degree\ polarization
l.tx_array.visualize;                           % Plot the array
l.rx_array = l.tx_array;                        % Use the same array for the Rx
```

Defining a track The third step defines the track. Here, we use a circle with 20 m diameter starting in the east, traveling north. We also choose a LOS scenario since we want to study the LOS polarization. The transmitter is located 8 m north of the center of the circle at an elevation of 2 m.

```
l.tx_position = [ 0 ; 12 ; 6 ];                 % Tx position
l.rx_position = [ 20 ; 0 ; 0 ];                 % Start position for the Rx track
l.track.generate('circular',40*pi,0);           % A circular track with radius 10 m
l.track.scenario = 'BERLIN_UMa_LOS';            % Chosse the Urban Macro LOS scenario
l.track.interpolate_positions( s.samples_per_meter );     % Interpolate positions
l.visualize;                                    % Plot the track
```

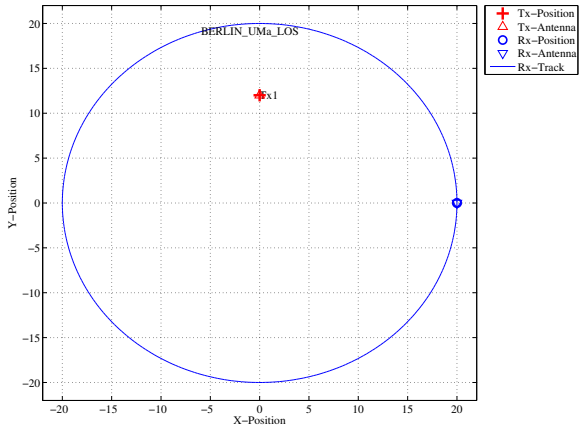

Figure 35: Scenario layout

Generating channel coefficients Now, we have finished the parametrization of the simulation and we can generate the parameters. We thus create a new set of correlated LSPs and fix the shadow fading and the K-factor to some default values. This disables the drifting for those parameters. We need to do that since otherwise, drifting and polarization would interfere with each other.

```
RandStream.setGlobalStream(RandStream('mt19937ar','seed',1));

p = l.create_parameter_sets;                    % Create parameter sets
p.parameter_maps(:,:,2) = 3;                    % Fix KF to 3 dB
p.parameter_maps(:,:,3) = 0;                    % Fix SF to 0 dB
p.plpar = [];                                   % Disable path loss model
p.update_parameters;

[c,cb] = p.get_channels;                        % Get the channel coefficients
```

```
Parameters    [ooooooooooooooooooooooooooooooooooooooooooooooooooooo]     1 seconds
Channels      [ooooooooooooooooooooooooooooooooooooooooooooooooooooo]     3 seconds
```

Results and Evaluation We now check the results and confirm, if they are plausible or not. We start with the two vertically polarized dipoles at the Tx and at the Rx side.

The model creates 8 taps, which is the default for the Urban-Macro LOS scenario. Without path-loss and shadow fading (SF=1), the power is normalized such that the sum over all taps is 1W and with a K-Factor of 3 dB, we get a received power of 0.67W for the LOS component. The remaining 0.33W are in the NLOS components. The results can be seen in Fig. 36 (top left).

```
figure                                              % New figure
plot(abs(squeeze( c.coeff(1,1,:,:) )').^2);         % Plot the graph
axis([0 360 -0.1 1]);                               % Set the axis
xlabel('Position [degrees]');                       % Add description
ylabel('LOS Power, linear scale');
title('Tx: Vertical , Rx: Vertical');               % Add title

disp(['LOS power:  ',num2str(mean( abs(c.coeff(1,1,1,:)).^2 , 4))])
disp(['NLOS power: ',num2str(mean( sum(abs(c.coeff(1,1,2:end,:)).^2,3) , 4))])
```

```
LOS power:  0.52851
NLOS power: 0.22249
```

The LOS power is almost constant when the Rx is south of the Tx. However, in close proximity (at 90°), the power is lowered significantly. This comes from the 2 m elevation of the Tx. When the Rx is almost under the Tx, the radiated power of the Dipole is much smaller compared to the broadside direction. The average power of the LOS is thus also lowered to 0.56 W. The average sum-power if the 7 NLOS components

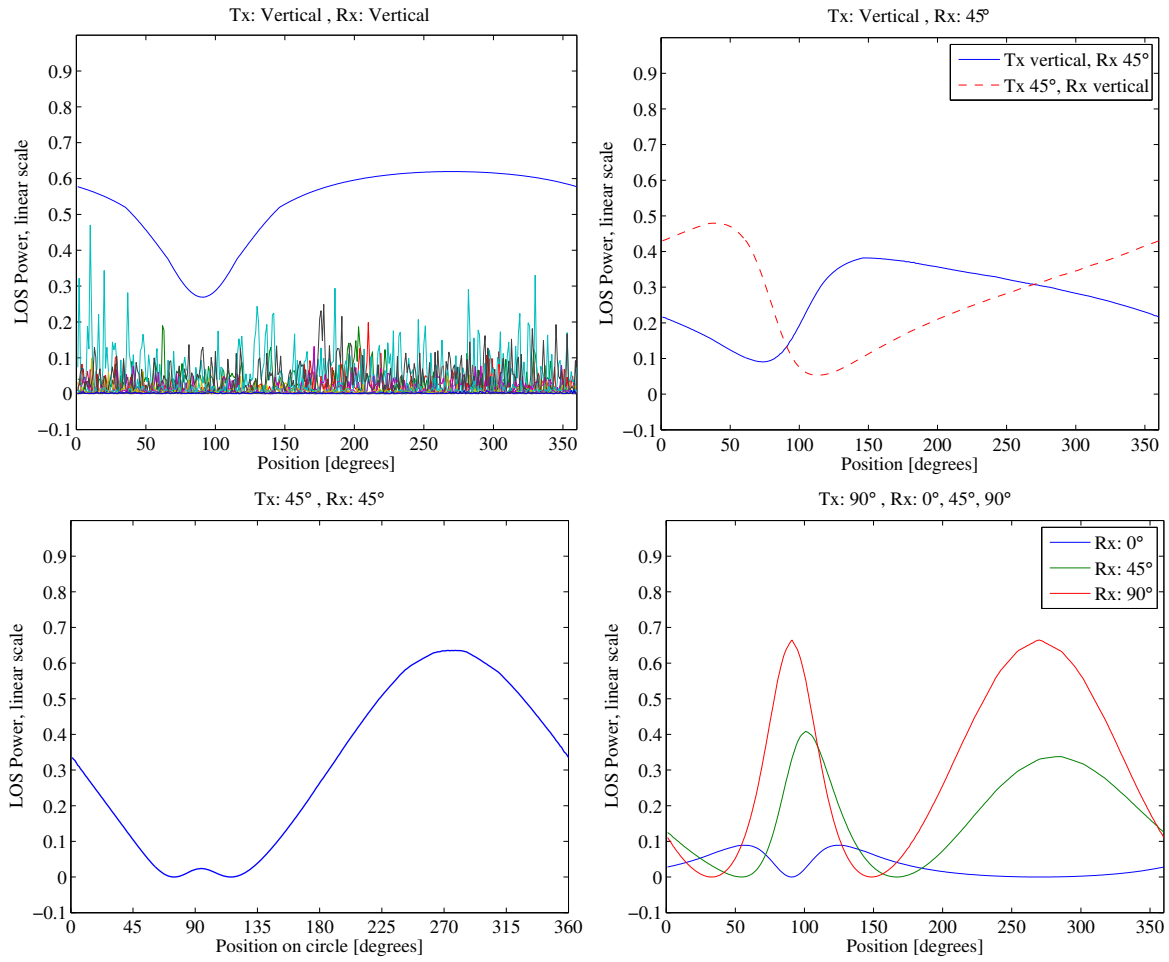

Figure 36: Results from the geometric polarization tutorial

is 0.26 W. This mainly come from the XPR which leakes some power from the vertical- into the horizontal polarization and thus reduces the received power on the vertically polarized Dipole.

Next, we study two cases. Either the Tx is vertical polarized and the Rx is at 45° or vise versa (Fig. 36, top right).

```
figure                                              % New figure
plot(abs(squeeze( c.coeff(2,1,1,:) )).^2);          % Tx vertical, Rx 45\degree\
hold on
plot(abs(squeeze( c.coeff(1,2,1,:) )).^2,'--r');    % Tx 45\degree\, Rx vertical
hold off
axis([0 360 -0.1 1]);
legend('Tx vertical, Rx 45\circ', 'Tx 45\circ, Rx vertical')
xlabel('Position [degrees]');
ylabel('LOS Power, linear scale');
title('Tx: Vertical , Rx: 45\circ');
```

The receiver changes its direction in a way that it always has the same orientation towards the Tx. However, due to the displacement of the Tx, the radiated power towards the Tx becomes minimal at around 90°. This minimum is visible in both curves (blue and red). However, the pole of the 45°slanted dipole now points to a different direction which explains the difference in the two lines. When the Rx is at 45°and the Tx is vertical, the pole is in the right half if the circle - resulting in a lower received power. When the Rx is Vertical and the Tx is 45°, the minimum power is achieved in the left half of the circle.

Next, we evaluate the two dipoles which are rotated by 45°. When moving around the circle, the Tx stays fixed and the Rx rotates. Subsequently, at one position, we will have both dipoles aligned and at another position, both will be crossed. When they are crossed, the received power will be 0 and when they are aligned, the power will match the first plot (two vertical dipoles). This can be seen in the following figure.

```
figure                                              % New figure
plot(abs(squeeze( c.coeff(2,2,1,:) )).^2 , 'Linewidth',1);
axis([0 360 -0.1 1]);
set(gca,'XTick',0:45:360)
xlabel('Position on circle [degrees]');
ylabel('LOS Power, linear scale');
title('Tx: 45\circ , Rx: 45\circ');
```

In the last figure, we rotated the transmit antenna by 90°. It is thus lying on the side and it is horizontally polarized. For the Rx, we consider three setups: Vertical (blue line), 45°(green line) and 90°(red line). Note that the Tx is rotated around the y-axis. At the initial position (0°), the Rx (45 and 90°) is rotated around the x-axis. This is because the movement direction.

```
figure                                              % New figure
plot(abs(squeeze( c.coeff(:,3,1,:) ))'.^2);
axis([0 360 -0.1 1]);
legend('Rx: 0\circ','Rx: 45\circ','Rx: 90\circ' )
xlabel('Position [degrees]');
ylabel('LOS Power, linear scale');
title('Tx: 90\circ , Rx: 0\circ, 45\circ, 90\circ');
```

When the receiver is vertical (blue line), both antennas are always crossed. There is no position around the circle where a good link can be established. When the receiver is horizontal (red line), however, there are two points where the two dipoles are aligned. For the 45°dipole, the same behavior can be observed but with roughly half the power.

```
close all
disp(['QuaDRiGa Version: ',simulation_parameters.version])
```

```
QuaDRiGa Version: 1.0.1-145
```

A.8 Visualizing RHCP/LHCP Patterns

The internal algorithms of the channel model only work with linear polarization. The antenna patterns are thus only stored in H/V polarization. This script defines two circular patch antennas and places them in an environment. It then rotates the transmit antenna degree by degree and thus samples all azimuth and elevation angles. The channel model is set up to record the channel response and thus record the RHCP/LHCP response like in a measurement in an anechoic chamber.

Set up the array We set up a patch antenna with an opening angle of 120° . We then copy that patch and rotate it by 90° around the x-axis to create an X-Pol array. We then set the coupling to +/- 90° phase to transmit circular polarized waves.

```
resolution = 10;        % in Degrees

a = array('custom',120,120,0);
a.set_grid( (-180:resolution:180)*pi/180 , (-90:resolution:90)*pi/180 );
a.copy_element(1,2);

b = a.copy_objects;

a.rotate_pattern(90,'x',2);
a.coupling = 1/sqrt(2) * [1 1;1j -1j];

b.rotate_pattern(90,'x',2);
b.coupling = 1/sqrt(2) * [1 1;1j -1j];
```

Place arrays in layout We place two of those arrays in a layout. The scenario 'LOSonly' has no NLOS scattering. One can see this setup as a perfect anechoic chamber.

```
l = layout;
l.simpar.show_progress_bars = 0;
l.simpar.drifting_precision = 0;

l.rx_position = [11;0;0];
l.track.no_snapshots = 1;
l.track.ground_direction = pi;
l.track.scenario = 'LOSonly';
l.tx_array = a;
l.rx_array = b;

p = l.create_parameter_sets;
[~,cb] = p.get_channels;
cb.pin = zeros( size(cb.pin) );
```

Get array response We now sample the array response for each degree in the antenna array.

```
pat = zeros( a.no_el , a.no_az , 2 , 2 );

values = a.no_az;
fprintf('Calculating  ['); m0=0; tStart = clock;  % A Status message
for n = 1:a.no_az
    m1=ceil(n/values*50); if m1>m0; for m2=1:m1-m0; fprintf('o'); end; m0=m1; end;

    a1 = a.copy_objects;
    a1.rotate_pattern( a.azimuth_grid(n)*180/pi , 'z');

    for m=1:a.no_el
        a2 = a1.copy_objects;
        a2.rotate_pattern( a.elevation_grid(m)*180/pi,'y');
        cb.tx_array = a2;
        c = cb.get_channels;
        pat(m,n,:,:) = c.coeff;
    end
end
fprintf('] %5.0f seconds\n',round( etime(clock, tStart) ));
```

```
Calculating  [oooooooooooooooooooooooooooooooooooooooooooooooooo]    17 seconds
```

Plot For plotting we use the internal function of the array class. We adjust the title of the figures accordingly.

```
d = a.copy_objects;
d.field_pattern_vertical =  pat(:,:,:,1) ;
d.field_pattern_horizontal = pat(:,:,:,2) ;
x = d.visualize;

set(x(1,11),'String','RHCP-RHCP')
set(x(1,12),'String','RHCP-LHCP')

set(x(2,11),'String','LHCP-RHCP')
set(x(2,12),'String','LHCP-LHCP')
```

```
close all
disp(['QuaDRiGa Version: ',simulation_parameters.version])
```

```
QuaDRiGa Version: 1.0.1-145
```

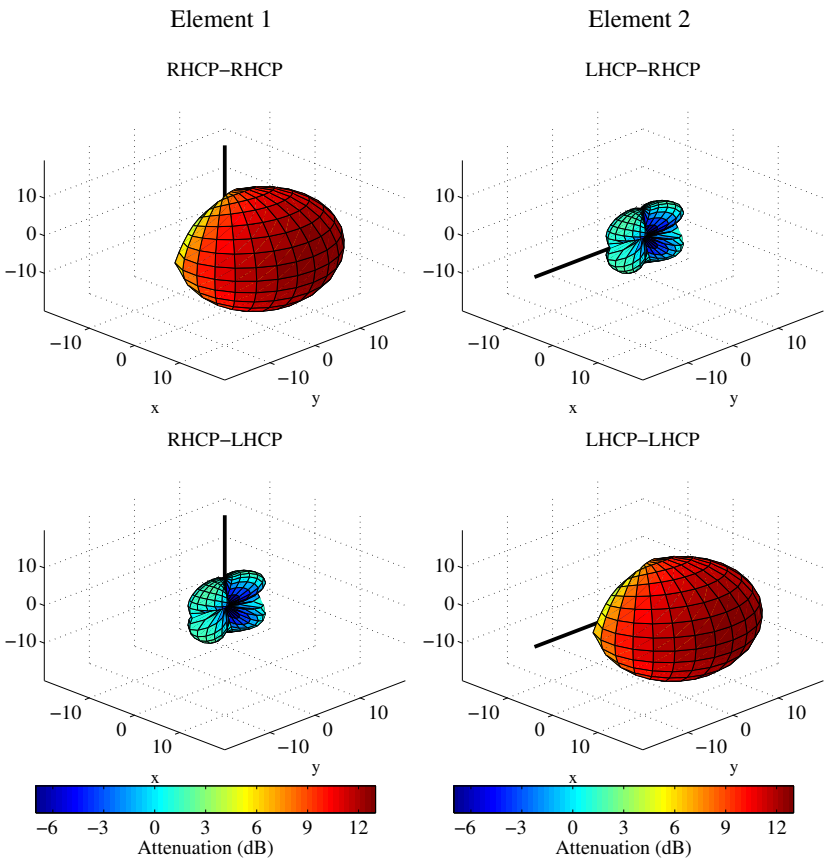

Figure 37: RHCP / LHCP antenna patterns

A.9 How to manually set LSPs in QuaDRiGa

This tutorial explains, how to generate a time-series of channel coefficients with manual selection of LSPs.

Setting general parameters We set up some basic parameters such as center frequency, sample density and the position of the transmitter.

```
close all
clear all

set(0,'defaultTextFontSize', 14)    % Set default font size for the plots
set(0,'defaultAxesFontSize', 14)

s = simulation_parameters;          % Basic simulation parameters
s.center_frequency      = 2.185e9;  % Center Frequency
s.sample_density        = 4;        % 4 samples / half wave length

l = layout(s);                      % Create Layout
```

Setting up a user track QuaDRiGa needs the positions of transmitter and receiver, e.g. for calculating the polarization or the arrival and departure angels. The positioning information of the Tx and Rx is essential also when the LSPs are not calculated. The following generates a linear track with 20 m length having a direction. The track is further split into 4 segments of 5 m length.

The splitting is done by calling the method 'split_segment' of the track object. The the first two arguments of that function are the minimum length of a segment (1 m) and the maximum length of the segment (6 m). Each existing segment that is longer then the maximum length is split into subsegments. The length of those segments is random where the third and fourth parameter determine the mean and the STD of the length of new subsegment. Hence, 't.split_segment(1,6,5,0)' splits all segment longer than 6 m into subsegments of 5 m length.

Each segment gets assigned a scenario. This is also essential since many parameters (such as the number of clusters, the XPR etc.) are scenario-specific. Hence, they are the same for the entire scenario. Here, we set the first the segments to NLOS, the third to LOS and the last to NLOS.

Last, we set a random starting position for the track in the layout.

```
l.tx_position = [0;0;25];                       % Set Tx-position

t = track('linear',20);                         % Linear track, 20 m length
t.interpolate_positions( s.samples_per_meter);  % Interpolate to sample density
t.split_segment( 1,6,5,0 )                      % Split in 4 segments

Un = 'WINNER_UMa_C2_NLOS';
Ul = 'WINNER_UMa_C2_LOS';
t.scenario = {Un,Un,Ul,Un};                     % Set scenarios

l.randomize_rx_positions( 500,0,0,0 );          % Random start position
tmp = l.rx_position;
l.track = t;
l.rx_position = tmp;

l.visualize;
```

Manual setting of the parameters Now we initialize the parameter-set objects.

The method "l.create_parameter_sets" splits the track into smaller sub-tracks, one for each segment. It further extracts the scenario informations. Each scenario gets its own parameter-set object. So we get an

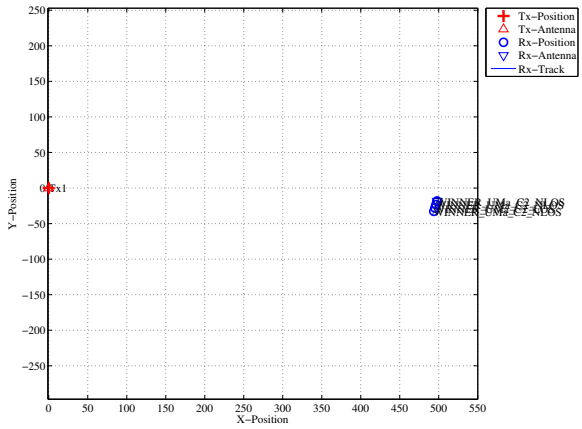

Figure 38: Scenario overview (manual parameter selection)

array of two parameter sets. The first element p(1) has three positions (NLOS segments) and the second has one position (LOS segment).

```
p = l.create_parameter_sets(1);

p(1).name
p(1).no_positions

p(2).name
p(2).no_positions
```

```
Parameters   [ooooooooooooooooooooooooooooooooooooooooooooooooooo]     5 seconds

ans =

WINNER-UMa-C2-NLOS_Tx1


ans =

     3


ans =

WINNER-UMa-C2-LOS_Tx1


ans =

     1
```

We set the number of clusters for the NLOS segments to 14. Currently, it is not possible to have a different number of clusters for each segment, i.e. it is not possible for the first NLOS segment to have 14 clusters and for the second to have only 10.

```
p(1).scenpar.NumClusters = 14;
```

In order to manually set the parameters, we have to overwrite the original settings. We do this here for the delay spread. The automatically generated values are:

```
[p(1).ds(1) p(1).ds(2) p(2).ds(1) p(1).ds(3) ]*1e6
```

```

ans =

    0.2696    0.2433    0.0948    0.2094
```

We want to set the values of the four segments to 0.45, 0.33, 0.12 and 0.60 microseconds. This is done by:

```
p(1).ds(1) = 0.45e-6;
p(1).ds(2) = 0.33e-6;
p(2).ds(1) = 0.12e-6;
p(1).ds(3) = 0.60e-6;
```

The K-Factor and the shadow fading are read from the map during the generation of channel coefficients. This would overwrite any manual values. However, this could be deactivated. A drawback is that in this case the KF, SF and PL are only updated once per segment. This will result in a step-like function of the output power. There is currently no method the set the power manually on a per-snapshot basis.

In the following example, we want to fix the received power to -102, -97, -82 and -99 dBm. K-Factors are taken from the map.

```
p(1).plpar = [];                        % Disable path loss for NLOS
p(2).plpar = [];                        % Disable path loss for LOS

p(1).sf = 10.^(0.1*[-102, -97, -99]);   % Set power for NLOS segments
p(2).sf = 10.^(0.1*[-82]);              % Set power for LOS segments

p(1).map_valid = false;                 % Disable automatic overwrite for NLOS
p(2).map_valid = false;                 % Disable automatic overwrite for LOS
```

Calculate channel coefficients Now we calculate the coefficients and the received power along the path. The following command calculate the channel coefficients. We then check the number of clusters that have been produced for each segment.

```
cm = p.get_channels;                    % Calculate the channel coefficients
cat( 1, cm.no_path )                    % Display the number of paths
```

```
Channels     [ooooooooooooooooooooooooooooooooooooooooooooooooooo]     7 seconds

ans =

    14
    14
    14
     8
```

The first three segments have 14 clusters. This has been set earlier. The last LOS segment has 15 clusters. One can see, that the three NLOS segment come first, followed by the LOS segment. The order has been scrambled. The following command sorts and combines the segments into one fading sequence.

```
c = cm.merge;
```

We now plot the power along the path. You can see the power levels of around -102, -97, -82 and -99 dBm which have been set earlier. Each segment has a transition area (e.g. from 2.5m to 5m, from 7.5m to 10m and from 12.5m to 15m) where the merging took place. In those areas, the power is scaled linearly. That means that, for example, in between 7.5 and 10m, the power ramps up from -97 to -82 dBm.

```
power = squeeze(sum(abs(c.coeff).^2,3));
power = 10*log10(power);

figure
[~,dist] = t.get_length;
plot(dist,power)
title('Simulated Power')
xlabel('Distance from start point [m]')
ylabel('Received Power [dBm]')
axis([0,20,-110,-80])
grid on
```

Figure 39: Power along the track (manual parameter selection)

The last plot shows the DS along the path. The results reflect the settings of 0.45, 0.33, 0.12 and 0.60 quiet well. As for the power, there is an overlap in between the segments. For example, in between 7.5 and 10m the DS drops from 0.33 to 0.12 microseconds. Additional fluctuations are caused by small scale fading.

```
coeff = squeeze( c.coeff );
delay = c.delay;

pow_tap = abs(coeff).^2;
pow_sum = sum(pow_tap);
mean_delay = sum( pow_tap.*delay) ./ pow_sum;
ds = sqrt( sum( pow_tap.*delay.^2)./ pow_sum - mean_delay.^2 );

figure
plot(dist,ds*1e6)
title('Simulated Delay Spread')
xlabel('Distance from start point [m]')
ylabel('RMS DS [\mus]')
axis([0,20,0,1])
grid on
```

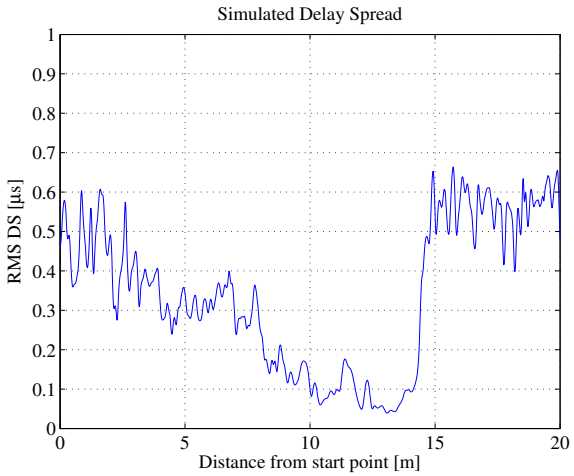

Figure 40: DS along the track (manual parameter selection)

```
close all
disp(['QuaDRiGa Version: ',simulation_parameters.version])
```

```
QuaDRiGa Version: 1.0.1-145
```